

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#)

Dokumentation zum Apache HTTP Server Version 2.2

Google-Suche

Hinweise zur Version

[Neue Funktionen in Version 2.1/2.2](#)

[Neue Funktionen in Version 2.0](#)

[Upgrade auf Version 2.2](#)

[Apache-Lizenz](#)

Bedienungsanleitung

[Kompilieren und Installieren](#)

[Starten](#)

[Stoppen und Neustarten](#)

[Konfigurationsanweisungen](#)

[Kurzreferenz der Direktiven](#)

[Module](#)

[Multi-Processing-Module \(MPMs\)](#)

[Filter](#)

[Handler](#)

[Server und Hilfsprogramme](#)

[Glossar](#)

Benutzerhandbuch

[Bindung](#)

[Konfigurationsdateien](#)

[Konfigurationsabschnitte](#)

[Caching von Inhalten](#)

[Content Negotiation](#)

[Dynamic Shared Objects \(DSO\)](#)

[Umgebungsvariablen](#)

[Log-Dateien](#)

[URLs auf das Dateisystem abbilden](#)

[Performance-Hinweise](#)

[Tipps zur Sicherheit](#)

[Serverweite Konfiguration](#)

[SSL/TLS-Verschlüsselung](#)

[Suexec für CGI](#)

[Einführung in die URL-Manipulation](#)

[Virtual-Hosts](#)

Praxis / Anleitungen

[Authentisierung und Autorisierung](#)

[Zugriffskontrolle](#)

[CGI: Dynamische Inhalte](#)

[.htaccess-Dateien](#)

[Server Side Includes \(SSI\)](#)

[Web-Verzeichnisse für Benutzer \(public_html\)](#)

Plattform-spezifische Anmerkungen

[Microsoft Windows](#)

[Novell NetWare](#)

[EBCDIC-Portierung](#)

Weitere Themen

[Häufig gestellte Fragen \(FAQ\)](#)

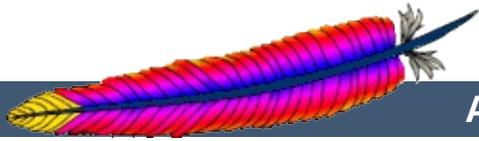
[Seitenindex](#)

[Dokumentation für Entwickler](#)

[Sonstiges](#)

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Upgrade von 2.0 auf 2.2

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Dieses Dokument dient der Unterstützung beim Upgrade. Es enthält die entscheidenden Informationen für bisherige Apache-Nutzer. Diese sind als kurze Anmerkungen gedacht. Weitere Informationen finden Sie entweder unter [Neue Funktionen](#) oder in den src/CHANGES-Dateien.

Dieses Dokument beschreibt lediglich die Änderungen von Version 2.2 gegenüber Version 2.0. Wenn Sie ein Upgrade von Version 1.3 durchführen, sollten Sie auch [Upgrade von 1.3 auf 2.0](#) zu Rate ziehen.

Siehe auch

[Übersicht der neuen Funktionen in Apache 2.0](#)



- Die APR 1.0 API ist erforderlich.
- Die mitgelieferte PCRE hat nun die Version 5.0.
- `mod_imap` wurde in `mod_imagemap` umbenannt.
- `mod_auth` wurde aufgeteilt in `mod_auth_basic` und `mod_authn_file`
- `mod_access` wurde umbenannt in `mod_authz_host`
- `mod_auth_ldap` wurde umbenannt in `mod_authz_ldap`



- Die mit dem Apache HTTP Server ausgelieferte Konfigurationsdatei `httpd.conf` wurde stark vereinfacht, indem alle außer den unbedingt notwendigen Konfigurationseinstellungen entfernt wurden. Im `conf/extra/`-Verzeichnis Ihrer Installation finden Sie eine Reihe von Konfigurationsbeispielen für erweiterte Funktionen.
- Die `apachectl startssl` gibt es nicht mehr. Um SSL-Unterstützung zu aktivieren müssen Sie die `httpd.conf` editieren und die entsprechenden `mod_ssl`-Anweisungen einfügen und anschließend den Server mit `apachectl start` starten. Eine Beispielkonfiguration zum Aktivieren von `mod_ssl` ist in `conf/extra/httpd-ssl.conf` enthalten.
- Die Voreinstellung von `UseCanonicalName` ist jetzt `Off`. Wenn diese Anweisung nicht in Ihrer Konfigurationsdatei enthalten ist, können Sie `UseCanonicalName On` einfügen, um das bisherige Verhalten beizubehalten.
- Das Modul `mod_userdir` reagiert nicht länger auf Anfragen, solange nicht in der Konfigurationsdatei eine `UserDir`-Anweisung enthalten ist, die einen Verzeichnisnamen angibt. Fügen Sie Ihrer Konfigurationsdatei die Anweisung `UserDir public_html` hinzu, um das bisherige Verhalten wiederherzustellen.

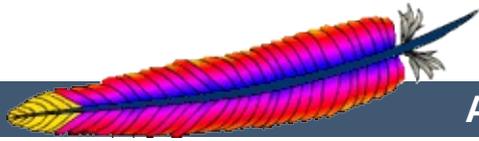


Change / Änderungen



Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Overview of new features in Apache 2.2

This document describes some of the major changes between the 2.0 and 2.2 versions of the Apache HTTP Server. For new features since version 1.3, see the [2.0 new features](#) document.



Authn/Authz

The bundled authentication and authorization modules have been refactored. The new [mod_auth_alias](#) module can greatly simplify certain authentication configurations. See [module name changes](#), and [the developer changes](#) for more information about how these changes affects users and module writers.

Caching

[mod_cache](#), [mod_disk_cache](#), and [mod_mem_cache](#) have undergone a lot of changes, and are now considered production-quality. [htcacheclean](#) has been introduced to clean up [mod_disk_cache](#) setups.

Configuration

The default configuration layout has been simplified and modularised. Configuration snippets which can be used to enable commonly-used features are now bundled with Apache, and can be easily added to the main server config.

Graceful stop

The [prefork](#), [worker](#) and [event](#) MPMs now allow [httpd](#) to be shutdown gracefully via the [graceful-stop](#) signal. The [GracefulShutdownTimeout](#) directive has been added to specify an optional timeout, after which [httpd](#) will terminate regardless of the status of any requests being served.

Proxying

The new [mod_proxy_balancer](#) module provides load balancing services for [mod_proxy](#). The new [mod_proxy_ajp](#) module adds support for the Apache JServ Protocol version 1.3 used by [Apache Tomcat](#).

Regular Expression Library Updated

Version 5.0 of the [Perl Compatible Regular Expression Library](#) (PCRE) is now included. [httpd](#) can be configured to use a system

installation of PCRE by passing the `--with-pcre` flag to configure.

Smart Filtering

`mod_filter` introduces dynamic configuration to the output filter chain. It enables filters to be conditionally inserted, based on any Request or Response header or environment variable, and dispenses with the more problematic dependencies and ordering problems in the 2.0 architecture.

Large File Support

`httpd` is now built with support for files larger than 2GB on modern 32-bit Unix systems. Support for handling >2GB request bodies has also been added.

Event MPM

The `event` MPM uses a separate thread to handle Keep Alive requests and accepting connections. Keep Alive requests have traditionally required `httpd` to dedicate a worker to handle it. This dedicated worker could not be used again until the Keep Alive timeout was reached.

SQL Database Support

`mod_dbd`, together with the `apr_dbd` framework, brings direct SQL support to modules that need it. Supports connection pooling in threaded MPMs.



Authn/Authz

Modules in the `aaa` directory have been renamed and offer better support for digest authentication. For example, `mod_auth` is now split into [`mod_auth_basic`](#) and [`mod_authn_file`](#); `mod_auth_dbm` is now called [`mod_authn_dbm`](#); `mod_access` has been renamed [`mod_authz_host`](#). There is also a new [`mod_authn_alias`](#) module for simplifying certain authentication configurations.

[`mod_authnz_ldap`](#)

This module is a port of the 2.0 `mod_auth_ldap` module to the 2.2 Authn/Authz framework. New features include using LDAP attribute values and complicated search filters in the [`Require`](#) directive.

[`mod_authz_owner`](#)

A new module that authorizes access to files based on the owner of the file on the file system

[`mod_version`](#)

A new module that allows configuration blocks to be enabled based on the version number of the running server.

[`mod_info`](#)

Added a new `?config` argument which will show the configuration directives as parsed by Apache, including their file name and line number. The module also shows the order of all request hooks and additional build information, similar to `httpd -V`.

[`mod_ssl`](#)

Added a support for [RFC 2817](#), which allows connections to upgrade from clear text to TLS encryption.

[`mod_imagemap`](#)

`mod_imap` has been renamed to [`mod_imagemap`](#) to avoid user confusion.



[httpd](#)

A new command line option `-M` has been added that lists all modules that are loaded based on the current configuration. Unlike the `-l` option, this list includes DSOs loaded via [mod_so](#).

[htttxt2dbm](#)

A new program used to generate dbm files from text input, for use in [RewriteMap](#) with the dbm map type.



APR 1.0 API

Apache 2.2 uses the APR 1.0 API. All deprecated functions and symbols have been removed from APR and APR-Util. For details, see the [APR Website](#).

Authn/Authz

The bundled authentication and authorization modules have been renamed along the following lines:

- `mod_auth_*` -> Modules that implement an HTTP authentication mechanism
- `mod_authn_*` -> Modules that provide a backend authentication provider
- `mod_authz_*` -> Modules that implement authorization (or access)
- `mod_authnz_*` -> Module that implements both authentication & authorization

There is a new authentication backend provider scheme which greatly eases the construction of new authentication backends.

Connection Error Logging

A new function, `ap_log_error` has been added to log errors that occur with the client's connection. When logged, the message includes the client IP address.

Test Configuration Hook Added

A new hook, `test_config` has been added to aid modules that want to execute special code only when the user passes `-t` to [httpd](#).

Set Threaded MPM's Stacksize

A new directive, [ThreadStackSize](#) has been added to set the stack size on all threaded MPMs. This is required for some third-party modules on platforms with small default thread stack size.

Protocol handling for output filters

In the past, every filter has been responsible for ensuring that it generates the correct response headers where it affects them. Filters can now delegate common protocol management to [mod_filter](#), using the `ap_register_output_filter_protocol` or `ap_filter_protocol` calls.

Monitor hook added

Monitor hook enables modules to run regular/scheduled jobs in the parent (root) process.

Regular expression API changes

The `pcregex.h` header is no longer available; it is replaced by the new `ap_regex.h` header. The POSIX.2 `regex.h` implementation exposed by the old header is now available under the `ap_` namespace from `ap_regex.h`. Calls to `regcomp`, `regexec` and so on can be replaced by calls to `ap_regcomp`, `ap_regexec`.

DBD Framework (SQL Database API)

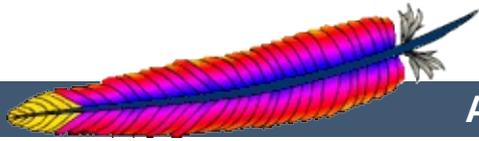
With Apache 1.x and 2.0, modules requiring an SQL backend had to take responsibility for managing it themselves. Apart from reinventing the wheel, this can be very inefficient, for example when several modules each maintain their own connections.

Apache 2.1 and later provides the `ap_dbd` API for managing database connections (including optimised strategies for threaded and unthreaded MPMs), while APR 1.2 and later provides the `apr_dbd` API for interacting with the database.

New modules SHOULD now use these APIs for all SQL database operations. Existing applications SHOULD be upgraded to use it where feasible, either transparently or as a recommended option to their users.

Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Übersicht der neuen Funktionen in Apache 2.0

Dieses Dokument beschreibt einige der wichtigsten Änderungen des Apache HTTP Servers 2.0 gegenüber der Version 1.3.

Siehe auch

[Upgrade von 1.3 auf 2.0](#)



Unix-Threading

Auf Unix-Systemen mit Unterstützung für POSIX-Threads, kann Apache jetzt in einem Multi-Process, Multi-Threaded Hybrid-Mode gestartet werden. Dies verbessert die Skalierfähigkeit für viele, jedoch nicht unbedingt alle Konfigurationen.

Neues Build-System

Das Build-System wurde komplett auf der Basis von autoconf und libtool neu geschrieben. Dadurch wird das Apache-Konfigurationssystem dem vieler anderer Packages ähnlicher.

Multi-Protokoll-Unterstützung

Apache stellt jetzt die notwendigen Grundfunktionalitäten bereit, um mehrere Protokolle unterstützen und verarbeiten zu können. [mod_echo](#) wurde hierfür als Beispiel geschrieben.

Bessere Unterstützung von Nicht-Unix-Plattformen

Apache 2.0 ist schneller und stabiler auf Nicht-Unix-Plattformen wie BeOS, OS/2 und Windows. Mit der Einführung von Plattformspezifischen [Multi-Processing Modulen](#) (MPMs) und der Apache Portable Runtime (APR), sind diese Plattformen jetzt in ihrem nativen API implementiert, wodurch die Verwendung der häufig fehlerbehafteten und schlecht funktionierenden POSIX-Emulation-Layer vermieden wird.

Neues Apache API

Das API für Module hat sich in 2.0 stark verändert. Die meisten der Sortierungs-/Prioritätsprobleme von Modulen bei 1.3 sollten nun verschwunden sein. In 2.0 wird hiervon vieles automatisch durchgeführt. Die Modulsortierung wird jetzt über einen pre-hook vorgenommen, um mehr Flexibilität zu bieten. Außerdem wurden neue API-Calls hinzugefügt, die zusätzliche Modulfähigkeiten zur Verfügung stellen, ohne den Apache-Kern anpassen zu müssen.

IPv6-Unterstützung

Auf Systemen, bei denen die zugrundeliegende Apache Portable

Runtime-Bibliothek IPv6 unterstützt, bekommt Apache standardmäßig IPv6 Listening Sockets. Zusätzlich unterstützen die Konfigurationsanweisungen [Listen](#), [NameVirtualHost](#) und [VirtualHost](#) numerische IPv6-Adressangaben (z.B., "Listen [2001:db8::1]:8080").

Filterung

Apache-Module können jetzt als Filter entwickelt und zur Filterung des rein- und rausgehenden Datenstroms des Servers eingesetzt werden. Hierdurch kann beispielsweise die Ausgabe von CGI-Skripten durch den INCLUDES-Filter von [mod_include](#) bearbeitet werden und so Server-Side Include-Anweisungen ausgeführt werden. Das Modul [mod_ext_filter](#) erlaubt externen Programmen als Filter zu agieren, in der gleichen Weise wie CGI-Programme als Eingabe dienen können.

Mehrsprachige Fehlermeldungen

Fehlermeldungen die an den Browser rausgehen, stehen jetzt als SSI-Dokumente in verschiedenen Sprachen zur Verfügung. Sie können bei Bedarf durch den Administrator angepasst werden, um ein einheitliches Design zu erreichen.

Vereinfachte Konfiguration

Viele der verwirrenden Konfigurationsanweisungen wurden vereinfacht. Die oft für Verwirrung sorgenden Port- und BindAddress-Anweisungen wurden entfernt. Ausschließlich die [Listen](#)-Anweisung wird nun zum Setzen von IP-Adressen und Portnummern benutzt. Der Servername und die Portnummer, die für Weiterleitungen und zur Erkennung virtueller Server verwendet werden, werden über die [ServerName](#)-Anweisung konfiguriert.

Native Windows NT Unicode-Unterstützung

Apache 2.0 auf Windows NT benutzt jetzt utf-8 für alle Dateinamen-Kodierungen. Diese werden direkt auf das zugrundeliegende Unicode-Dateisystem abgebildet, wodurch Mehrsprach-Unterstützung für alle Windows NT-basierten Installationen, inklusive

Windows 2000 und Windows XP, zur Verfügung gestellt wird. *Diese Unterstützung ist nicht auf Windows 95, 98 oder ME verfügbar. Hier wird weiterhin die jeweils lokale Codepage des Rechners für den Zugriff auf das Dateisystem verwendet.*

Bibliothek für reguläre Ausdrücke aktualisiert

Apache 2.0 enthält die ["Perl Compatible Regular Expression Library"](#) (PCRE). Bei der Auswertung aller regulären Ausdrücke wird nun die leistungsfähigere Syntax von Perl 5 verwendet.



mod_ssl

Neues Modul in Apache 2.0. Dieses Modul ist ein Interface zu den von OpenSSL bereitgestellten SSL/TLS Verschlüsselungs-Protokollen.

mod_dav

Neues Modul in Apache 2.0. Dieses Modul implementiert die HTTP Distributed Authoring and Versioning (DAV) Spezifikation zur Erzeugung und Pflege von Web-Inhalten.

mod_deflate

Neues Modul in Apache 2.0. Dieses Modul erlaubt es Browsern, die dies unterstützen, eine Komprimierung des Inhaltes vor der Auslieferung anzufordern, um so Netzwerk-Bandbreite zu sparen.

mod_auth_ldap

Neues Modul in Apache 2.0.41. Diese Modul ermöglicht die Verwendung einer LDAP-Datenbank zur Speicherung von Berechtigungsdaten für die HTTP-Basic-Authentication. Ein Begleitmodul, [mod_ldap](#), stellt einen Verbindungs-Pool und die Pufferung von Abfrageergebnissen zur Verfügung.

mod_auth_digest

Zusätzliche Unterstützung für prozessübergreifendes Session-Caching mittels Shared-Memory.

mod_charset_lite

Neues Modul in Apache 2.0. Dieses experimentelle Modul erlaubt Zeichensatz-Übersetzungen oder -Umschlüsselung.

mod_file_cache

Neues Modul in Apache 2.0. Dieses Modul beinhaltet die Funktionalität von `mod_mmap_static` aus Apache 1.3, plus einige weitere Caching-Funktionen.

mod_headers

Dieses Modul ist in Apache 2.0 deutlich flexibler geworden. Es kann

jetzt die von [mod_proxy](#) genutzten Request-Header manipulieren und es ist möglich Response-Header auf Basis von definierten Bedingungen zu verändern.

mod_proxy

Das Proxy Modul wurde komplett neu geschrieben um die Möglichkeiten der neuen Filter-Funktionalität auszuschöpfen und um einen zuverlässigen Proxy zu haben, der den HTTP/1.1-Spezifikationen entspricht. Neue [<Proxy>](#)-Konfigurationsabschnitte bieten eine besser lesbare (und intern schnellere) Kontrolle der vermittelten Seiten. Die überladenen `<Directory "proxy: . . .">`-Konfigurationen werden nicht mehr unterstützt. Das Modul ist nun in mehrere Module unterteilt, die jeweils ein bestimmtes Übertragungsprotokoll unterstützen, wie `proxy_connect`, `proxy_ftp` und `proxy_http`.

mod_negotiation

Die neue Konfigurationsanweisung [ForceLanguagePriority](#) kann benutzt werden, um sicherzustellen, dass ein Client auf jeden Fall ein einzelnes Dokument, anstatt einer NOT ACCEPTABLE- oder MULTIPLE CHOICES-Antwort, bekommt. Zusätzlich wurden die Negotiation- und Multiview-Algorithmen angepasst um einheitlichere Ergebnisse zu liefern. Außerdem wird ein neues Type-Map-Format bereitgestellt, das Dokumenteninhalte direkt enthalten kann.

mod_autoindex

Automatisch erzeugte Verzeichnisindizes können zur besseren Übersichtlichkeit durch HTML-Tabellen dargestellt werden. Genauere Sortierungen, wie Sortierung nach Versionsnummer und Wildcard-Filterung des Verzeichnisindizes werden unterstützt.

mod_include

Neue Anweisungen erlauben es, die Standard Start- und Endtags von SSI-Elementen zu ändern. Zudem können die Default-Formate für Fehlermeldungen und Zeitangaben nun ebenfalls in der Serverkonfiguration vorgenommen werden. Auf die Ergebnisse der

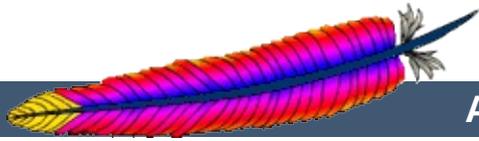
Auswertung und Gruppierung von regulären Ausdrücken (jetzt auf Basis der Perl-Syntax für reguläre Ausdrücke) kann über die [mod_include](#) Variablen \$0 bis \$9 zugegriffen werden.

mod_auth_dbm

DBM-ähnliche Datenbanken werden jetzt durch die Konfigurationsweisung `AuthDBMType` unterstützt.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

The Apache License, Version 2.0

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not

limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by

Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. **Grant of Patent License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. **Redistribution.** You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - a. You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - b. You must cause any modified files to carry prominent notices stating that You changed the files; and

- c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions.** Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall

supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. **Trademarks.** This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. **Disclaimer of Warranty.** Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity,

or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

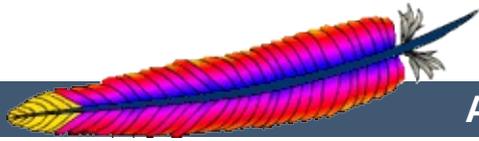
```
Copyright [yyyy] [name of copyright owner]
```

```
Licensed under the Apache License, Version 2.0 (the "License");  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at
```

```
http://www.apache.org/licenses/LICENSE-2.0
```

```
Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or impl  
See the License for the specific language governing permissions and  
limitations under the License.
```

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Kompilieren und Installieren

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Dieses Dokument umfaßt nur die Kompilierung und Installation des Apache HTTP Servers auf Unix und Unix-ähnlichen Systemen. Für die Kompilierung und Installation unter Windows lesen Sie bitte [Den Apache HTTPd unter Microsoft Windows betreiben](#). Für andere Plattformen lesen Sie bitte die Dokumentation [Plattformen](#).

Der Apache HTTPd verwendet `libtool` und `autoconf`, um eine Erstellungsumgebung zu schaffen, die der vieler anderer Open Source Projekte ähnlich sieht.

Wenn Sie von einer Unterversion auf die nächste aktualisieren (z.B. von 2.2.50 auf 2.2.51), springen Sie bitte zum Abschnitt [Upgrade](#).

Siehe auch

[Den Quellcode konfigurieren](#)

[Den Apache HTTP Server starten](#)

[Beenden und Neustarten](#)



```
Download      $ lynx
                http://httpd.apache.org/download.cgi
Auspacken    $ gzip -d httpd-NN.tar.gz
                $ tar xvf httpd-NN.tar $ cd httpd-NN
Konfigurieren $ ./configure --prefix=PREFIX
Kompilieren  $ make
Installieren  $ make install
Anpassen     $ vi PREFIX/conf/httpd.conf
Testen       $ PREFIX/bin/apachectl -k start
```

NN muss durch die entsprechende Versionsnummer ersetzt werden und *PREFIX* durch den Verzeichnispfad, in dem der Server installiert werden soll. Wenn *PREFIX* nicht angegeben ist, wird die Voreinstellung `/usr/local/apache2` verwendet.

Beginnend mit den Anforderungen für die Kompilierung und Installation des Apache HTTP Servers ist weiter unten jeder Abschnitt des Kompilierungs- und Installationsvorganges genauer beschrieben.



Folgende Anforderungen gelten für die Erstellung des Apache HTTPd:

Plattenplatz

Stellen Sie sicher, dass Sie kurzzeitig wenigstens 50 MB freien Festplattenplatz zur Verfügung haben. Nach der Installation belegt der Apache ungefähr 10 MB Plattenplatz. Der tatsächliche Platzbedarf variiert in Abhängigkeit von den gewählten Konfigurationseinstellungen und Modulen von Drittanbietern.

ANSI-C-Compiler und Generierungswerkzeuge

Stellen Sie sicher, dass Sie einen ANSI-C Compiler installiert haben. Der [GNU C Compiler \(GCC\)](#) der [Free Software Foundation \(FSF\)](#) ist empfehlenswert. Wenn Sie den GCC nicht besitzen, stellen Sie zumindest sicher, dass der Compiler Ihres Anbieters ANSI-kompatibel ist. Außerdem muss Ihr PATH wesentliche Generierungswerkzeuge wie make enthalten.

Zeitgenauigkeit bewahren

Elemente des HTTP-Protokolls werden in Form einer Tageszeit ausgedrückt. Darum sollten Sie jetzt prüfen, ob Ihr System die Fähigkeit zur Zeitsynchronisation besitzt, und diese gegebenenfalls installieren. Üblicherweise werden hierfür die Programme `ntpd` oder `xntpd` verwendet, die auf dem Network Time Protocol (NTP) basieren. Nähere Informationen über NTP Software und öffentliche Zeitserver finden Sie auf der [NTP-Homepage](#).

[Perl 5](#) [OPTIONAL]

Für einige Hilfsskripte wie [apxs](#) oder [dbmmanage](#) (die in Perl geschrieben sind) wird der Perl 5 Interpreter benötigt (die Versionen ab 5.003 sind ausreichend). Wenn Sie mehrere Perl Interpreter haben (beispielsweise eine systemweite Installation von Perl 4 und Ihre eigene Perl 5-Installation), dann sollten Sie die `--with-perl-` Option (siehe unten) verwenden, um sicherzustellen, dass der richtige Interpreter von [configure](#) ausgewählt wird. Wenn kein Perl 5-Interpreter vom [configure](#)-Skript gefunden werden kann, können

Sie die betroffenen Hilfsskripte nicht verwenden, sind jedoch selbstverständlich nach wie vor in der Lage, den Apache HTTPd zu bauen und zu installieren.

apr/apr-util >= 1.2

apr und apr-util werden mit den Quelltext-Veröffentlichungen des Apache HTTPd gebündelt und in nahezu allen Fällen ohne Probleme verwendet. Sind jedoch die apr- oder apr-util-Versionen 1.0 oder 1.1 auf Ihrem System installiert, dann müssen Sie diese apr-/apr-util-Installationen entweder aktualisieren oder httpd separate Kompilate verwenden lassen. Um hierfür die mitgelieferten apr-/apr-util-Quellen zu verwenden, müssen Sie diese manuell installieren:

```
# apr 1.2 bauen und installieren
cd srclib/apr
./configure --prefix=/usr/local/apr-httpd/
make
make install

# apr-util 1.2 bauen und installieren
cd ../apr-util
./configure --prefix=/usr/local/apr-util-httpd/ --with-
apr=/usr/local/apr-httpd/
make
make install

# httpd konfigurieren
cd ../../
./configure --with-apr=/usr/local/apr-httpd/ --with-apr-
util=/usr/local/apr-util-httpd/
```



Der Apache HTTP Server kann von der [Apache-HTTP-Server-Downloadseite](#) heruntergeladen werden, auf der verschiedene Spiegelserver angegeben sind. Für die meisten Benutzer des Apache HTTPd ist es auf Unix-ähnlichen Systemen am Besten, die Quellcodeversion herunterzuladen und zu kompilieren. Der Erstellungsprozess (weiter unten beschrieben) ist einfach und erlaubt es Ihnen, den Server Ihren Bedürfnissen anzupassen. Dazu kommt, dass Binärdistributionen gegenüber der aktuellen Quellcodeversion oft veraltet sind. Wenn Sie tatsächlich ein Binärpaket herunterladen, folgen Sie bitte den Anweisungen in der Datei `INSTALL.bindist`, die der Distribution beiliegt.

Es ist wichtig, dass Sie nach dem Herunterladen überprüfen, dass es sich um eine vollständige und unveränderte Version des Apache HTTP Servers handelt. Das können Sie erreichen, indem Sie das heruntergeladene Paket gegen die PGP-Signatur prüfen. Einzelheiten dazu erfahren Sie auf der [Download-Seite](#). Es ist auch ein erweitertes Beispiel verfügbar, das die [Anwendung von PGP](#) beschreibt.



Auspacken

Das Auspacken des Quellcodes aus dem Apache-HTTPd-Tarball besteht aus einem simplen Dekomprimieren und danach "Ent-tarren":

```
$ gzip -d httpd-NN.tar.gz  
$ tar xvf httpd-NN.tar
```

Dies erstellt unterhalb des aktuellen Verzeichnisses ein neues Verzeichnis, das den Quellcode für die Distribution enthält. Sie sollten mit `cd` in dieses Verzeichnis wechseln, bevor Sie mit der Kompilierung des Servers weitermachen.



Der nächste Schritt ist die Konfiguration des Apache-HTTPd-Codebaumes für Ihre spezielle Plattform und Ihre persönlichen Bedürfnisse. Dies wird mit dem Skript [configure](#) durchgeführt, das im Wurzelverzeichnis der Distribution enthalten ist. (Entwickler, welche eine nicht freigegebene Version des Apache-HTTPd-Codebaumes heruntergeladen, müssen `autoconf` und `libtool` installiert haben und müssen `buildconf` ausführen, bevor sie mit den nächsten Schritten fortfahren können. Dies wird bei offiziellen Releases nicht notwendig sein.)

Um den Codebaum mit den Standardeinstellungen zu konfigurieren, geben Sie einfach `./configure` ein. Zur Änderung dieser Voreinstellungen akzeptiert [configure](#) eine Reihe von Variablen und Kommandozeilenoptionen.

Die wichtigste Option ist `--prefix`, der Ablageort, an dem der Apache HTTP Server später installiert wird, da er für diesen Ort konfiguriert werden muss, um korrekt zu arbeiten. Eine feinere Einstellung der Dateiablagen ist mit weiteren [configure-Optionen](#) möglich.

Weiterhin können Sie zu diesem Zeitpunkt festlegen, welche [Funktionalität](#) Sie in den Apache HTTPd aufnehmen möchten, indem Sie [Module](#) aktivieren oder deaktivieren. Der Apache HTTP Server bindet standardmäßig einen Satz von [Basismodulen](#) ein. Andere Module werden mit Hilfe der Option `--enable-module` aktiviert, wobei *module* den Namen des Moduls ohne das Präfix `mod_` darstellt. Ausserdem sind alle Unterstriche durch Bindestriche zu ersetzen. Sie können sich auch entscheiden, Module als "[Shared Objects \(DSOs\)](#)" zu kompilieren, welche zur Laufzeit ge- und entladen werden können. Dazu verwenden Sie die Option `--enable-module=shared`. Entsprechend können Sie Basismodule mit der Option `--disable-module` deaktivieren. Lassen Sie Vorsicht walten, wenn Sie diese Optionen verwenden, da [configure](#) Sie nicht warnen kann, wenn die von Ihnen angegebenen

Module nicht existieren; die Option wird dann einfach ignoriert.

Zusätzlich ist es zuweilen notwendig, das [configure](#)-Skript mit Extraintformationen zum Ablageort Ihres Compilers, Ihrer Bibliotheken oder Header-Dateien zu versorgen. Das tun Sie, indem Sie entweder Umgebungsvariablen oder Kommandozeilenoptionen an [configure](#) übergeben. Für mehr Informationen lesen Sie bitte die Hilfeseite zu [configure](#).

Um einen kurzen Eindruck zu gewinnen, welche Möglichkeiten Sie haben, folgt hier ein typisches Beispiel, das den Apache mit einem speziellen Compiler und Compilerflags für das Installationsverzeichnis `/sk/pkg/apache` kompiliert, sowie die beiden zusätzlichen Module [mod_rewrite](#) und [mod_speling](#) für späteres Laden durch den DSO-Mechanismus:

```
$ CC="pgcc" CFLAGS="-O2" \  
./configure --prefix=/sw/pkg/apache \  
--enable-rewrite=shared \  
--enable-speling=shared
```

Wenn [configure](#) startet, benötigt es mehrere Minuten, um die Verfügbarkeit von Features auf Ihrem System zu prüfen und ein Makefile zu generieren, das später zur Kompilierung des Servers verwendet wird.

Einzelheiten zu den vielen verschiedenen [configure](#)-Optionen finden Sie auf der Hilfeseite zu [configure](#).



Nun können Sie die verschiedenen Teile, die das Apache-HTTPd-Paket bilden, einfach durch Ausführen des folgenden Befehls erstellen:

```
$ make
```

Seien Sie hierbei bitte geduldig, denn eine Basiskonfiguration benötigt mehrere Minuten zum Kompilieren. Die Zeit kann jedoch abhängig von Ihrer Hardware und der Anzahl der Module, die Sie aktiviert haben, sehr stark variieren.



Nun endlich installieren Sie das Package unter dem konfigurierten Installations-*PREFIX* (siehe oben: Option `--prefix` durch Aufrufen von:

```
$ make install
```

Wenn Sie upgraden, wird die Installation Ihre Konfigurationsdateien oder Dokumente nicht überschrieben.



Als nächstes können Sie Ihren Apache HTTP Server anpassen, indem Sie die [Konfigurationsdateien](#) unterhalb von `PREFIX/conf/` editieren.

```
$ vi PREFIX/conf/httpd.conf
```

Werfen Sie auch einen Blick in das Apache-HTTP-Server-Handbuch unter [docs/manual/](#). Die aktuellste Version dieses Handbuchs sowie eine komplette Referenz der verfügbaren [Konfigurationsanweisungen](#) finden Sie unter <http://httpd.apache.org/docs/2.2/>.



Sie können nun Ihren Apache HTTP Server [starten](#), indem Sie einfach

```
$ PREFIX/bin/apachectl -k start
```

ausführen.

Danach sollten Sie Ihr erstes Dokument unter dem URL `http://localhost/` anfordern können. Die Webseite, die Sie sehen, ist im [DocumentRoot](#) abgelegt, welches üblicherweise `PREFIX/htdocs/` ist. Den Server [stoppen](#) Sie wieder durch Ausführen von:

```
$ PREFIX/bin/apachectl -k stop
```



Der erste Schritt beim Aktualisieren besteht darin, die Versionsankündigung sowie die CHANGES-Datei in der Quelltextdistribution zu lesen, um Änderungen zu finden, die Ihr System möglicherweise betreffen. Wenn Sie einen größeren Versionsprung durchführen (z.B. vom 1.3 auf 2.0 oder von 2.0 auf 2.2), wird es wahrscheinlich auch größere Unterschiede in der Kompilier- und Laufzeitkonfiguration geben, die manuelle Nacharbeiten erfordern. Außerdem müssen alle Module aktualisiert werden, um den Änderungen der Modul-API gerecht zu werden.

Die Aktualisierung einer Unterversion auf eine andere (z.B. von 2.2.55 auf 2.2.57) ist einfacher. `make install` überschreibt keine der bereits existierenden Dokumente, Log- und Konfigurationsdateien. Ausserdem bemühen sich die Entwickler, inkompatible Änderungen der `configure`-Optionen, der Laufzeitkonfiguration sowie der Modul-API zu vermeiden. In den meisten Fällen sollten Sie in der Lage sein, den gleichen `configure`-Befehl, die gleiche Konfiguration und die gleichen Module wieder zu verwenden.

Um auf eine neue Unterversion zu aktualisieren, suchen Sie zunächst die Datei `config.nice` im `build`-Verzeichnis Ihrer Serverinstallation oder im Wurzelverzeichnis des Quelltextbaums der alten Installation. Die Datei enthält den genauen `configure`-Befehl, der verwendet wurde, um den Quellcode zu konfigurieren. Um jetzt von einer Version auf die nächste zu aktualisieren, kopieren Sie einfach die `config.nice` in das Verzeichnis der neuen Version, passen sie bei Bedarf an, und führen Sie sie aus:

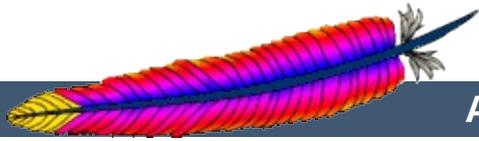
```
$ ./config.nice
$ make
$ make install
$ PREFIX/bin/apachectl -k graceful-stop
$ PREFIX/bin/apachectl -k start
```

Sie sollten jede neue Version immer in Ihrer Umgebung testen, bevor

Sie sie produktiv schalten. Beispielsweise können Sie die neue Version neben der alten installieren, indem Sie ein anderes `--prefix` und einen anderen Port wählen (durch Anpassen der `Listen`-Direktive). So können Sie auf eventuelle Inkompatibilitäten testen, bevor Sie endgültig die neue Version verwenden.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Apache starten

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Unter Windows läuft der Apache üblicherweise als Dienst (Windows NT, 2000 und XP) oder als Konsolenanwendung (Windows 9x und ME). Für Einzelheiten lesen Sie bitte [Apache als Dienst betreiben](#) und [Apache als Konsolenanwendung betreiben](#).

Unter Unix wird das [httpd](#)-Programm als Daemon ausgeführt, der im Hintergrund fortlaufend aktiv ist, um Anfragen zu bearbeiten. Dieses Dokument beschreibt, wie [httpd](#) aufgerufen wird.

Siehe auch

[Beenden und Neustarten](#)

[httpd](#)

[apachectl](#)



Wie der Apache startet

Wenn die in der Konfigurationsdatei angegebene [Listen](#)-Anweisung auf die Voreinstellung von 80 gesetzt ist (oder einen anderen Port unterhalb von 1024), dann müssen Sie root-Berechtigung besitzen, um den Apache starten zu können. Nur dann kann er sich an diesen privilegierten Port binden. Sobald der Server gestartet ist und einige vorbereitende Aktionen wie das Öffnen seiner Log-Dateien ausgeführt hat, startet er mehrere *Kind*-Prozesse, welche die Arbeit erledigen: das Lauschen auf und Beantworten von Anfragen von Clients. Der Haupt-`httpd`-Prozess läuft unter dem Benutzer `root` weiter, die *Kind*-Prozesse jedoch werden unter weniger privilegierten Benutzerkennungen ausgeführt. Dies wird von dem ausgewählten [Multi-Processing-Modul](#) gesteuert.

Die Verwendung des Steuerskripts [apachectl](#) ist die empfohlene Methode, das `httpd`-Programm zu starten. Dieses Skript setzt verschiedene Umgebungsvariablen, die für die korrekte Funktion von `httpd` unter einigen Betriebssystemen notwendig sind, und startet dann das `httpd`-Programm. [apachectl](#) reicht alle Kommandozeilenargumente durch, so dass alle `httpd`-Optionen auch mit [apachectl](#) verwendet werden können. Um den korrekten Ablageort des `httpd`-Programms sowie einige Kommandozeilenargumente anzugeben, die Sie *immer* verwenden möchten, können Sie auch das Skript [apachectl](#) direkt editieren und die Variable `HTTPD` am Anfang ändern.

Das Erste was `httpd` macht, wenn es startet, ist das Suchen und Einlesen der [Konfigurationsdatei](#) `httpd.conf`. Der Ablageort dieser Datei wird zur Kompilierungszeit festgelegt. Es ist aber möglich, den Ablageort zur Laufzeit anzugeben, indem die Kommandozeilenoption `-f` wie folgt verwendet wird:

```
/usr/local/apache2/bin/apachectl -f /usr/local/apache2/conf/httpd.conf
```

Wenn während des Starts alles gutgeht, trennt sich der Server vom

Terminal ab und die Eingabeaufforderung erscheint gleich darauf wieder. Dies zeigt an, dass der Server hochgefahren ist und läuft. Sie können nun Ihren Browser benutzen, um Verbindung zum Server aufzunehmen und sich die Testseite im [DocumentRoot](#)-Verzeichnis anzusehen wie auch die lokale Kopie der Dokumentation, die von dieser Seite aus verlinkt ist.



Wenn der Apache während des Hochfahrens einen schweren Fehler feststellt, schreibt er entweder eine Nachricht, die das Problem näher schildert, auf die Konsole oder ins [ErrorLog](#), bevor er sich selbst beendet. Eine der häufigsten Fehlermeldungen ist "Unable to bind to Port ... " (*Anm.d.Ü.:* "Kann nicht an Port ... binden"). Diese Meldung wird üblicherweise verursacht:

- entweder durch den Versuch, den Server an einem privilegierten Port zu starten, während man nicht als Benutzer root angemeldet ist,
- oder durch den Versuch, den Server zu starten, wenn bereits eine andere Instanz des Apache oder ein anderer Webserver an den gleichen Port gebunden ist.

Für weitere Anleitungen zur Fehlerbehebung lesen Sie bitte die [Apache-FAQ](#).



Den Bootvorgang starten

Wenn Sie möchten, dass Ihr Server direkt nach einem System-Neustart weiterläuft, sollten Sie einen Aufruf von [apachectl](#) zu den Startdateien Ihres Systems hinzufügen (üblicherweise `rc.local` oder eine Datei in einem `rc.N`-Verzeichnis). Dies startet den Apache als `root`. Stellen Sie zuvor jedoch sicher, dass Ihr Server hinsichtlich Sicherheit und Zugriffsbeschränkungen richtig konfiguriert ist.

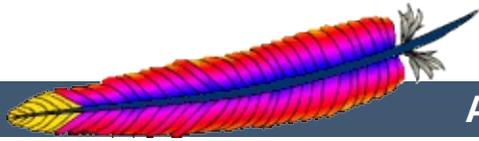
Das [apachectl](#)-Skript ist dafür ausgelegt, wie ein Standard-SysV-init-Skript zu arbeiten. Es akzeptiert die Argumente `start`, `restart` und `stop` und übersetzt sie in die entsprechenden Signale für [httpd](#). Daher können Sie oftmals einfach [apachectl](#) in das entsprechende init-Verzeichnis linken. Überprüfen Sie bitte auf jeden Fall die genauen Anforderungen Ihres Systems.



Weitere Informationen über Kommandozeilenoptionen von [httpd](#) und [apachectl](#) sowie anderen Hilfsprogrammen, die dem Server beigelegt sind, sind auf der Seite [Server und Hilfsprogramme](#) verfügbar. Es existiert außerdem eine Dokumentation aller in der Apache-Distribution enthaltenen [Module](#) und der von ihnen bereitgestellten [Direktiven](#).

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Beenden und Neustarten

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Dieses Dokument umfasst das Beenden und Neustarten des Apache auf Unix-ähnlichen Systemen. Anwender von Windows NT, 2000 und XP sollten [Betreiben des Apache als Dienst](#) lesen, während hingegen Anwender von Windows 9x sowie ME [Betreiben des Apache als Konsolenanwendung](#) lesen sollten, um mehr Informationen zur Handhabung des Apache auf diesen Systemen zu erhalten.

Siehe auch

[httpd](#)
[apachectl](#)



Um den Apache zu stoppen oder neu zu starten, müssen Sie ein Signal an den laufenden [httpd](#)-Prozess senden. Es gibt zwei Möglichkeiten, diese Signale zu senden. Zum einen können Sie den Unix-Befehl `kill` verwenden, um den Prozessen direkt Signale zu senden. Sie werden feststellen, dass auf Ihrem System mehrere [httpd](#)-Programme laufen. Sie sollten jedoch nicht jedem dieser Prozesse ein Signal senden, sondern nur dem Elternprozess, dessen PID im [PidFile](#) steht. Das heißt, Sie sollten es niemals nötig haben, einem anderen Prozess, als dem Elternprozess, ein Signal zu senden. Es gibt drei Signale, die Sie an den Elternprozess senden können: [TERM](#), [HUP](#) und [USR1](#), die nachfolgend beschrieben werden.

Um dem Elternprozess ein Signal zu senden, verwenden Sie einen Befehl wie z.B.:

```
kill -TERM `cat /usr/local/apache2/logs/httpd.pid`
```

Die zweite Methode, dem [httpd](#)-Prozess zu signalisieren, ist die Verwendung der `-k`-Befehlszeilenoptionen `stop`, `restart`, `graceful` und `graceful-stop`, wie unten beschrieben. Dies sind Argumente des [httpd](#)-Programms, es wird jedoch empfohlen, sie unter Verwendung des Steuerskripts [apachectl](#) zu senden, welches diese an [httpd](#) durchreicht.

Nachdem Sie [httpd](#) signalisiert haben, können Sie dessen Fortschritt beobachten, indem Sie eingeben:

```
tail -f /usr/local/apache2/logs/error_log
```

Passen Sie diese Beispiele entsprechend Ihren [ServerRoot](#)- und [PidFile](#)-Einstellungen an.



Signal: TERM

```
apachectl -k stop
```

Das Senden des TERM- oder stop-Signals an den Elternprozess veranlasst diesen, sofort zu versuchen, alle seine Kindprozesse zu beenden. Es kann einige Sekunden dauern, bis alle Kindprozesse komplett beendet sind. Danach beendet sich der Elternprozess selbst. Alle gerade bearbeiteten Anfragen werden abgebrochen. Es werden keine weiteren Anfragen mehr bedient.



Signal: USR1

```
apachectl -k graceful
```

Das USR1- oder graceful-Signal veranlasst den Elternprozess, die Kinder *anzuweisen*, sich nach Abschluß ihrer momentanen bearbeiteten Anfrage zu beenden (oder sich sofort zu beenden, wenn sie gerade keine Anfrage bedienen). Der Elternprozess liest seine Konfigurationsdateien erneut ein und öffnet seine Logdateien neu. Wenn ein Kindprozess stirbt, ersetzt der Elternprozess ihn durch ein Kind der neuen Konfigurations-*Generation*. Dieses beginnt sofort damit, neue Anfragen zu bedienen.

Der Code ist dafür ausgelegt, stets die MPM-Direktiven zur Prozesssteuerung zu beachten, so dass die Anzahl der Prozesse und Threads, die zur Bedienung der Clients bereitstehen, während des Neustarts auf die entsprechenden Werte gesetzt werden. Weiterhin wird StartServers auf folgende Art und Weise interpretiert: Wenn nach einer Sekunde nicht mindestens StartServers neue Kindprozesse erstellt wurden, dann werden, um den Durchsatz zu beschleunigen, entsprechend weitere erstellt. Auf diese Weise versucht der Code sowohl die Anzahl der Kinder entsprechend der Serverlast anzupassen als auch Ihre Wünsche hinsichtlich des Parameters StartServers zu berücksichtigen.

Benutzer von mod_status werden feststellen, dass die Serverstatistiken **nicht** auf Null zurückgesetzt werden, wenn ein USR1 gesendet wurde. Der Code wurde so geschrieben, dass sowohl die Zeit minimiert wird, in der der Server nicht in der Lage ist, neue Anfragen zu bedienen (diese werden vom Betriebssystem in eine Warteschlange gestellt, so dass sie auf keinen Fall verloren gehen) als auch Ihre Parameter zur Feinabstimmung berücksichtigt werden. Um dies zu erreichen, muss die *Statusabelle* (Scoreboard), die dazu verwendet wird, alle Kinder über mehrere Generationen zu verfolgen, erhalten bleiben.

Das Statusmodul benutzt außerdem ein G, um diejenigen Kinder zu kennzeichnen, die noch immer Anfragen bedienen, welche gestartet wurden, bevor ein unterbrechungsfreier Neustart veranlaßt wurde.

Derzeit gibt es keine Möglichkeit für ein Log-Rotationsskript, das USR1 verwendet, sicher festzustellen, dass alle Kinder, die in ein vor dem Neustart geöffnetes Log schreiben, beendet sind. Wir schlagen vor, dass Sie nach dem Senden des Signals USR1 eine angemessene Zeitspanne warten, bevor Sie das alte Log anfassen. Wenn beispielsweise die meisten Ihrer Zugriffe bei Benutzern mit niedriger Bandbreite weniger als 10 Minuten für eine vollständige Antwort benötigen, dann könnten Sie 15 Minuten warten, bevor Sie auf das alte Log zugreifen.

Wenn Ihre Konfigurationsdatei Fehler enthält, während Sie einen Neustart anweisen, dann wird Ihr Elternprozess nicht neu starten, sondern sich mit einem Fehler beenden. Im Falle eines unterbrechungsfreien Neustarts läßt er die Kinder weiterlaufen, wenn er sich beendet. (Dies sind die Kinder, die sich "sanft beenden", indem sie ihre letzte Anfrage erledigen.) Das verursacht Probleme, wenn Sie versuchen, den Server neu zu starten -- er ist nicht in der Lage, sich an die Ports zu binden, an denen er lauschen soll. Bevor Sie einen Neustart durchführen, können Sie die Syntax der Konfigurationsdateien mit dem Befehlszeilenargument `-t` überprüfen (siehe auch [httpd](#)). Das garantiert allerdings nicht, dass der Server korrekt starten wird. Um sowohl die Syntax als auch die Semantik der Konfigurationsdateien zu prüfen, können Sie versuchen, [httpd](#) als nicht-root-Benutzer zu starten. Wenn dabei keine Fehler auftreten, wird er versuchen, seine Sockets und Logdateien zu öffnen und fehlschlagen, da er nicht root ist (oder weil sich der gegenwärtig laufende [httpd](#) bereits diese Ports gebunden hat). Wenn er aus einem anderen Grund fehlschlägt, dann liegt wahrscheinlich ein Konfigurationsfehler vor. Der Fehler sollte behoben werden, bevor der unterbrechungsfreie Neustart angewiesen wird.



Signal: HUP

```
apachectl -k restart
```

Das Senden des Signals HUP oder `restart` veranlaßt den Elternprozess, wie bei TERM alle seine Kinder zu beenden. Der Elternprozess beendet sich jedoch nicht. Er liest seine Konfigurationsdateien neu ein und öffnet alle Logdateien erneut. Dann erzeugt er einen neuen Satz Kindprozesse und setzt die Bedienung von Zugriffen fort.

Benutzer von [mod_status](#) werden feststellen, dass die Serverstatistiken auf Null gesetzt werden, wenn ein HUP gesendet wurde.

Wenn Ihre Konfigurationsdatei einen Fehler enthält, während Sie einen Neustart anweisen, dann wird Ihr Elternprozess nicht neu starten, sondern sich mit einem Fehler beenden. Lesen Sie oben, wie Sie das vermeiden können.



Signal: WINCH

```
apachectl -k graceful stop
```

Das WINCH- oder graceful-stop-Signal veranlasst den Elternprozess, die Kinder *anzuweisen*, sich nach Abschluß ihrer momentan bearbeiteten Anfrage zu beenden (oder sich sofort zu beenden, wenn sie gerade nichts bedienen). Der Elternprozess entfernt dann sein [PidFile](#) und stellt das Lauschen auf allen Ports ein. Er läuft weiter und beobachtet alle Kindprozesse, die noch Anfragen bearbeiten. Sobald alle Kindprozesse fertig sind und beendet haben oder die mit [GracefulShutdownTimeout](#) definierte Zeitüberschreitung erreicht wurde, beendet sich der Elternprozess ebenfalls. Jedem verbliebenen Kindprozess wird beim Erreichen der Zeitüberschreitung das TERM-Signal gesendet, um diesen zum Beenden zu zwingen.

Ein TERM-Signal beendet den Elternprozess und alle Kindprozesse unverzüglich, wenn sie sich im "graceful"-Status (*Anm.d.Ü.: wörtl. "gnädiger" Status*) befinden. Da jedoch das [PidFile](#) dann schon gelöscht ist, werden Sie dieses Signal nicht mehr mit `apachectl` oder `ht tpd` senden können.

Das Signal graceful-stop ermöglicht Ihnen den Betrieb mehrerer identisch konfigurierter Instanzen von `ht tpd` zur gleichen Zeit. Dies ist eine mächtige Funktionalität bei der Aufrüstung des Apache. Sie kann jedoch bei einigen Konfigurationen auch zur gegenseitigen Blockierung und zu Wettlaufsituationen führen.

Es ist besonders darauf zu achten, dass auf Festplatte gespeicherte Dateien wie [Lockfile](#) und [ScriptSock](#) die Server-PID enthalten und ohne Probleme nebeneinander existieren müssen. Wann auch immer eine Konfigurationsanweisung, ein Drittanbieter-Modul oder ein persistentes CGI-Skript irgend eine Sperre oder eine Statusdatei auf Festplatte speichert, muss besonders darauf geachtet werden, dass

mehrere gleichzeitig laufende Instanzen von `httpd` sich nicht gegenseitig die Dateien zerstören.

Sie sollten ebenfalls vorsichtig mit möglichen Wettlaufsituationen sein, wie beispielsweise der Verwendung von weitergeleiteter Protokollierung nach der Art von [rotatelogs](#). Mehrere gleichzeitig laufende Instanzen von [rotatelogs](#), die versuchen, die gleichen Protokolldateien zu rotieren, können sich gegenseitig die Protokolldateien zerstören.



Vor der Version 1.2b9 des Apache existierten verschiedene *Wettlaufsituationen* (*Anm.d.Ü.:* engl.: *race conditions*), die den Neustart und die Signale beeinflusst haben (einfach gesagt, eine Wettlaufsituation ist ein zeitabhängiges Problem - wenn etwas zum falschen Zeitpunkt oder in der falschen Reihenfolge geschieht, kommt es zu nicht erwünschten Ergebnissen. Geschehen die gleichen Dinge zur rechten Zeit, ist alles in Ordnung). Bei Architekturen mit dem "richtigen" (*Anm.d.Ü.:* im Sinne von "geeignet") Funktionsumfang haben wir so viele eliminiert wie wir nur konnten. Dennoch sollte beachtet werden, dass noch immer Wettlaufsituationen auf bestimmten Architekturen existieren.

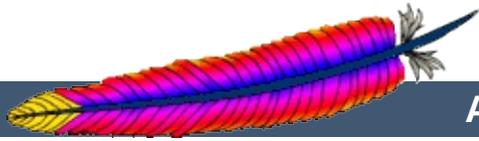
Bei Architekturen, die ein [ScoreBoardFile](#) auf Platte verwenden, kann die Statustabelle beschädigt werden. Das kann zu "bind: Address already in use" ("bind: Adresse wird bereits verwendet", nach einem HUP) oder "long lost child came home!" ("Der verlorene Sohn ist heimgekehrt", nach einem USR1) führen. Ersteres ist ein schwerer Fehler, während letzteres lediglich bewirkt, dass der Server einen Eintrag in der Statustabelle verliert. So kann es ratsam sein, unterbrechungsfreie Neustarts zusammen mit einem gelegentlichen harten Neustart zu verwenden. Diese Probleme lassen sich nur sehr schwer umgehen, aber glücklicherweise benötigen die meisten Architekturen keine Statustabelle in Form einer Datei. Bitte lesen Sie für Architekturen, die sie benötigen, die Dokumentation zu [ScoreBoardFile](#).

Alle Architekturen haben in jedem Kindprozess eine kleine Wettlaufsituation, welche die zweite und nachfolgende Anfragen einer persistenten HTTP-Verbindung (KeepAlive) umfaßt. Der Prozess kann nach dem Lesen der Anfragezeile aber vor dem Lesen der Anfrage-Header enden. Es existiert eine Korrektur, die für 1.2 zu spät kam. Theoretisch sollte das kein Problem darstellen, da der KeepAlive-Client derartige Ereignisse aufgrund von Netzwerk-Latenzzeiten und Auszeiten des Servers erwarten sollte. In der Praxis scheint keiner von beiden beeinflusst zu werden -- in einem Testfall wurde der Server zwanzig mal

pro Sekunde neu gestartet, während Clients das Angebot abgegrast haben, ohne kaputte Bilder oder leere Dokumente zu erhalten.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Konfigurationsdateien

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Dieses Dokument beschreibt die Dateien, die zur Konfiguration des Apache HTTP Servers verwendet werden.



Referenzierte Module	Referenzierte Direktiven
<u>mod_mime</u>	<u><IfDefine></u> <u>Include</u> <u>TypesConfig</u>

Der Apache wird konfiguriert, indem Direktiven in einfache Textdateien eingetragen werden. Die Hauptkonfigurationsdatei heißt üblicherweise `httpd.conf`. Der Ablageort dieser Datei wird bei der Kompilierung festgelegt, kann jedoch mit der Befehlszeilenoption `-f` überschrieben werden. Durch Verwendung der Direktive Include können außerdem weitere Konfigurationsdateien hinzugefügt werden. Zum Einfügen von mehreren Konfigurationsdateien können Platzhalter verwendet werden. Jede Direktive darf in jeder dieser Konfigurationsdateien angegeben werden. Änderungen in den Hauptkonfigurationsdateien werden vom Apache nur beim Start oder Neustart erkannt.

Der Server liest auch eine Datei mit MIME-Dokumenttypen ein. Der Name dieser Datei wird durch die Direktive TypesConfig bestimmt. Die Voreinstellung ist `mime.types`.



Syntax der Konfigurationsdateien

Die Konfigurationsdateien des Apache enthalten eine Direktive pro Zeile. Der Backslash "\" läßt sich als letztes Zeichen in einer Zeile dazu verwenden, die Fortsetzung der Direktive in der nächsten Zeile anzuzeigen. Es darf kein weiteres Zeichen oder Whitespace zwischen dem Backslash und dem Zeilenende folgen.

In den Konfigurationsdateien wird bei den Direktiven nicht zwischen Groß- und Kleinschreibung unterschieden. Bei den Argumenten der Direktiven wird dagegen oftmals zwischen Groß- und Kleinschreibung differenziert. Zeilen, die mit dem Doppelkreuz "#" beginnen, werden als Kommentare betrachtet und ignoriert. Kommentare dürfen **nicht** am Ende einer Zeile nach der Direktive eingefügt werden. Leerzeilen und Whitespaces vor einer Direktive werden ignoriert. Dadurch lassen sich Direktiven zur besseren Lesbarkeit einrücken.

Sie können die Syntax Ihrer Konfigurationsdateien auf Fehler prüfen, ohne den Server zu starten, indem Sie `apachectl configtest` oder die Befehlszeilenoption `-t` verwenden.



Referenzierte Module

mod_so

Referenzierte Direktiven

<IfModule>

LoadModule

Der Apache ist ein modularer Server. Das bedeutet, dass nur die absolute Grundfunktionalität im Kernserver enthalten ist. Weitergehende Fähigkeiten sind mittels [Modulen](#) verfügbar, die in den Apache geladen werden können. Standardmäßig wird bei der Kompilierung ein Satz von Basismodulen (*Anm.d.Ü.:* die so genannten [Base](#)-Module) in den Server eingebunden. Wenn der Server für die Verwendung von [dynamisch ladbaren](#) Modulen kompiliert wurde, dann können Module separat kompiliert und jederzeit mittels der Direktive [LoadModule](#) hinzugefügt werden. Andernfalls muss der Apache neu kompiliert werden, um Module hinzuzufügen oder zu entfernen. Konfigurationsanweisungen können abhängig vom Vorhandensein eines bestimmten Moduls eingesetzt werden, indem sie in einen [<IfModule>](#)-Block eingeschlossen werden.

Um zu sehen, welche Module momentan in den Server einkompiliert sind, kann die Befehlszeilenoption `-l` verwendet werden.



Referenzierte Module

Referenzierte Direktiven

[<Directory>](#)
[<DirectoryMatch>](#)
[<Files>](#)
[<FilesMatch>](#)
[<Location>](#)
[<LocationMatch>](#)
[<VirtualHost>](#)

Direktiven in den Hauptkonfigurationsdateien gelten für den gesamten Server. Wenn Sie die Konfiguration nur für einen Teil des Servers verändern möchten, können Sie den Gültigkeitsbereich der Direktiven beschränken, indem Sie diese in [<Directory>](#)-, [<DirectoryMatch>](#)-, [<Files>](#)-, [<FilesMatch>](#)-, [<Location>](#)- oder [<LocationMatch>](#)-Abschnitte eingefügen. Diese Abschnitte begrenzen die Anwendung der umschlossenen Direktiven auf bestimmte Pfade des Dateisystems oder auf bestimmte URLs. Sie können für eine fein abgestimmte Konfiguration auch ineinander verschachtelt werden.

Der Apache besitzt die Fähigkeit, mehrere verschiedene Websites gleichzeitig zu bedienen. Dies wird [virtuelles Hosten](#) genannt. Direktiven können auch in ihrem Gültigkeitsbereich eingeschränkt werden, indem sie innerhalb eines [<VirtualHost>](#)-Abschnittes angegeben werden. Sie werden dann nur auf Anfragen für eine bestimmte Website angewendet.

Obwohl die meisten Direktiven in jedem dieser Abschnitte platziert werden können, ergeben einige Direktiven in manchen Kontexten keinen Sinn. Direktiven zur Prozesssteuerung beispielsweise dürfen nur im Kontext des Hauptservers angegeben werden. Prüfen Sie den [Kontext](#) der Direktive, um herauszufinden, welche Direktiven in welche Abschnitte eingefügt werden können. Weitere Informationen finden Sie unter "[Wie](#)

[Directory-, Location- und Files-Abschnitte arbeiten](#)".



Referenzierte Module

Referenzierte Direktiven

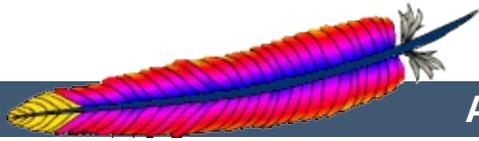
[AccessFileName](#)

[AllowOverride](#)

Der Apache ermöglicht die dezentrale Verwaltung der Konfiguration mittels spezieller Dateien innerhalb des Web-Verzeichnisbaums. Diese speziellen Dateien heißen gewöhnlich `.htaccess`, mit der Direktive [AccessFileName](#) kann jedoch auch ein anderer Name festgelegt werden. In `.htaccess`-Dateien angegebene Direktiven werden auf das Verzeichnis und dessen Unterverzeichnisse angewendet, in dem die Datei abgelegt ist. `.htaccess`-Dateien folgen der gleichen Syntax wie die Hauptkonfigurationsdateien. Da `.htaccess`-Dateien bei jeder Anfrage eingelesen werden, werden Änderungen in diesen Dateien sofort wirksam.

Prüfen Sie den [Kontext](#) der Direktive, um herauszufinden, welche Direktiven in `.htaccess`-Dateien angegeben werden können. Darüber hinaus steuert der Serveradministrator mit der Einstellung der Direktive [AllowOverride](#) in den Hauptkonfigurationsdateien welche Direktiven in `.htaccess`-Dateien verwendet werden dürfen.

Weitere Informationen über `.htaccess`-Dateien finden Sie in der [.htaccess-Einführung](#).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Configuration Sections

Directives in the [configuration files](#) may apply to the entire server, or they may be restricted to apply only to particular directories, files, hosts, or URLs. This document describes how to use configuration section containers or `.htaccess` files to change the scope of other configuration directives.



Related Modules	Related Directives
core	<Directory>
mod_version	<DirectoryMatch>
mod_proxy	<Files>
	<FilesMatch>
	<IfDefine>
	<IfModule>
	<IfVersion>
	<Location>
	<LocationMatch>
	<Proxy>
	<ProxyMatch>
	<VirtualHost>

There are two basic types of containers. Most containers are evaluated for each request. The enclosed directives are applied only for those requests that match the containers. The [<IfDefine>](#), [<IfModule>](#), and [<IfVersion>](#) containers, on the other hand, are evaluated only at server startup and restart. If their conditions are true at startup, then the enclosed directives will apply to all requests. If the conditions are not true, the enclosed directives will be ignored.

The [<IfDefine>](#) directive encloses directives that will only be applied if an appropriate parameter is defined on the [httpd](#) command line. For example, with the following configuration, all requests will be redirected to another site only if the server is started using `httpd -DClosedForNow`:

```
<IfDefine ClosedForNow>
  Redirect / http://otherserver.example.com/
</IfDefine>
```

The `<IfModule>` directive is very similar, except it encloses directives that will only be applied if a particular module is available in the server. The module must either be statically compiled in the server, or it must be dynamically compiled and its `LoadModule` line must be earlier in the configuration file. This directive should only be used if you need your configuration file to work whether or not certain modules are installed. It should not be used to enclose directives that you want to work all the time, because it can suppress useful error messages about missing modules.

In the following example, the `MimeMagicFile` directive will be applied only if `mod_mime_magic` is available.

```
<IfModule mod_mime_magic.c>
    MimeMagicFile conf/magic
</IfModule>
```

The `<IfVersion>` directive is very similar to `<IfDefine>` and `<IfModule>`, except it encloses directives that will only be applied if a particular version of the server is executing. This module is designed for the use in test suites and large networks which have to deal with different httpd versions and different configurations.

```
<IfVersion >= 2.1>
    # this happens only in versions greater or
    # equal 2.1.0.
</IfVersion>
```

`<IfDefine>`, `<IfModule>`, and the `<IfVersion>` can apply negative conditions by preceding their test with "!". Also, these sections can be nested to achieve more complex restrictions.



The most commonly used configuration section containers are the ones that change the configuration of particular places in the filesystem or web space. First, it is important to understand the difference between the two. The filesystem is the view of your disks as seen by your operating system. For example, in a default install, Apache resides at `/usr/local/apache2` in the Unix filesystem or `"c:/Program Files/Apache Group/Apache2"` in the Windows filesystem. (Note that forward slashes should always be used as the path separator in Apache, even for Windows.) In contrast, the web space is the view of your site as delivered by the web server and seen by the client. So the path `/dir/` in the web space corresponds to the path `/usr/local/apache2/htdocs/dir/` in the filesystem of a default Apache install on Unix. The web space need not map directly to the filesystem, since webpages may be generated dynamically from databases or other locations.

Filesystem Containers

The `<Directory>` and `<Files>` directives, along with their [regex](#) counterparts, apply directives to parts of the filesystem. Directives enclosed in a `<Directory>` section apply to the named filesystem directory and all subdirectories of that directory (as well as the files in those directories). The same effect can be obtained using [.htaccess files](#). For example, in the following configuration, directory indexes will be enabled for the `/var/web/dir1` directory and all subdirectories.

```
<Directory /var/web/dir1>
    Options +Indexes
</Directory>
```

Directives enclosed in a `<Files>` section apply to any file with the specified name, regardless of what directory it lies in. So for example, the following configuration directives will, when placed in the main section of

the configuration file, deny access to any file named `private.html` regardless of where it is found.

```
<Files private.html>
  Order allow,deny
  Deny from all
</Files>
```

To address files found in a particular part of the filesystem, the [<Files>](#) and [<Directory>](#) sections can be combined. For example, the following configuration will deny access to `/var/web/dir1/private.html`, `/var/web/dir1/subdir2/private.html`, `/var/web/dir1/subdir3/private.html`, and any other instance of `private.html` found under the `/var/web/dir1/` directory.

```
<Directory /var/web/dir1>
  <Files private.html>
    Order allow,deny
    Deny from all
  </Files>
</Directory>
```

Webpace Containers

The [<Location>](#) directive and its [regex](#) counterpart, on the other hand, change the configuration for content in the webpace. For example, the following configuration prevents access to any URL-path that begins in `/private`. In particular, it will apply to requests for `http://yoursite.example.com/private`, `http://yoursite.example.com/private123`, and `http://yoursite.example.com/private/dir/file.html` as well as any other requests starting with the `/private` string.

```
<LocationMatch ^/private>
  Order Allow,Deny
  Deny from all
</LocationMatch>
```

The `<Location>` directive need not have anything to do with the filesystem. For example, the following example shows how to map a particular URL to an internal Apache handler provided by `mod_status`. No file called `server-status` needs to exist in the filesystem.

```
<Location /server-status>
  SetHandler server-status
</Location>
```

Wildcards and Regular Expressions

The `<Directory>`, `<Files>`, and `<Location>` directives can each use shell-style wildcard characters as in `fnmatch` from the C standard library. The character `"*` matches any sequence of characters, `"?"` matches any single character, and `"[seq]"` matches any character in `seq`. The `"/"` character will not be matched by any wildcard; it must be specified explicitly.

If even more flexible matching is required, each container has a regular expression (regex) counterpart `<DirectoryMatch>`, `<FilesMatch>`, and `<LocationMatch>` that allow perl-compatible [regular expressions](#) to be used in choosing the matches. But see the section below on configuration merging to find out how using regex sections will change how directives are applied.

A non-regex wildcard section that changes the configuration of all user directories could look as follows:

```
<Directory /home/*/public_html>
```

```
Options Indexes
</Directory>
```

Using regex sections, we can deny access to many types of image files at once:

```
<FilesMatch \.(?i:gif|jpe?g|png)$>
    Order allow,deny
    Deny from all
</FilesMatch>
```

What to use When

Choosing between filesystem containers and webpace containers is actually quite easy. When applying directives to objects that reside in the filesystem always use [<Directory>](#) or [<Files>](#). When applying directives to objects that do not reside in the filesystem (such as a webpage generated from a database), use [<Location>](#).

It is important to never use [<Location>](#) when trying to restrict access to objects in the filesystem. This is because many different webpace locations (URLs) could map to the same filesystem location, allowing your restrictions to be circumvented. For example, consider the following configuration:

```
<Location /dir/>
    Order allow,deny
    Deny from all
</Location>
```

This works fine if the request is for `http://yoursite.example.com/dir/`. But what if you are on a case-insensitive filesystem? Then your restriction could be easily circumvented by requesting `http://yoursite.example.com/DIR/`.

The `<Directory>` directive, in contrast, will apply to any content served from that location, regardless of how it is called. (An exception is filesystem links. The same directory can be placed in more than one part of the filesystem using symbolic links. The `<Directory>` directive will follow the symbolic link without resetting the pathname. Therefore, for the highest level of security, symbolic links should be disabled with the appropriate `Options` directive.)

If you are, perhaps, thinking that none of this applies to you because you use a case-sensitive filesystem, remember that there are many other ways to map multiple web space locations to the same filesystem location. Therefore you should always use the filesystem containers when you can. There is, however, one exception to this rule. Putting configuration restrictions in a `<Location />` section is perfectly safe because this section will apply to all requests regardless of the specific URL.



VIRTUAL HOSTS

The `<VirtualHost>` container encloses directives that apply to specific hosts. This is useful when serving multiple hosts from the same machine with a different configuration for each. For more information, see the [Virtual Host Documentation](#).



The `<Proxy>` and `<ProxyMatch>` containers apply enclosed configuration directives only to sites accessed through `mod_proxy`'s proxy server that match the specified URL. For example, the following configuration will allow only a subset of clients to access the `www.example.com` website using the proxy server:

```
<Proxy "http://www.example.com/*">  
    Order allow,deny  
    Allow from 192.168.1.104 192.168.1.205  
    Deny from all  
</Proxy>
```



To find out what directives are allowed in what types of configuration sections, check the [Context](#) of the directive. Everything that is allowed in [<Directory>](#) sections is also syntactically allowed in [<DirectoryMatch>](#), [<Files>](#), [<FilesMatch>](#), [<Location>](#), [<LocationMatch>](#), [<Proxy>](#), and [<ProxyMatch>](#) sections. There are some exceptions, however:

- The [AllowOverride](#) directive works only in [<Directory>](#) sections.
- The `FollowSymLinks` and `SymLinksIfOwnerMatch` [Options](#) work only in [<Directory>](#) sections or `.htaccess` files.
- The [Options](#) directive cannot be used in [<Files>](#) and [<FilesMatch>](#) sections.



How the sections are merged

The configuration sections are applied in a very particular order. Since this can have important effects on how configuration directives are interpreted, it is important to understand how this works.

The order of merging is:

1. `<Directory>` (except regular expressions) and `.htaccess` done simultaneously (with `.htaccess`, if allowed, overriding `<Directory>`)
2. `<DirectoryMatch>` (and `<Directory ~>`)
3. `<Files>` and `<FilesMatch>` done simultaneously
4. `<Location>` and `<LocationMatch>` done simultaneously

Apart from `<Directory>`, each group is processed in the order that they appear in the configuration files. `<Directory>` (group 1 above) is processed in the order shortest directory component to longest. So for example, `<Directory /var/web/dir>` will be processed before `<Directory /var/web/dir/subdir>`. If multiple `<Directory>` sections apply to the same directory they are processed in the configuration file order. Configurations included via the `Include` directive will be treated as if they were inside the including file at the location of the `Include` directive.

Sections inside `<VirtualHost>` sections are applied *after* the corresponding sections outside the virtual host definition. This allows virtual hosts to override the main server configuration.

When the request is served by `mod_proxy`, the `<Proxy>` container takes the place of the `<Directory>` container in the processing order.

Technical Note

There is actually a `<Location>/<LocationMatch>` sequence

performed just before the name translation phase (where `Aliases` and `DocumentRoots` are used to map URLs to filenames). The results of this sequence are completely thrown away after the translation has completed.

Relationship between modules and configuration sections

One question that often arises after reading how configuration sections are merged is related to how and when directives of specific modules like `mod_rewrite` are processed. The answer is not trivial and needs a bit of background. Each `httpd` module manages its own configuration, and each of its directives in `httpd.conf` specify one piece of configuration in a particular context. `httpd` does not execute a command as it is read.

At runtime, the core of `httpd` iterates over the defined configuration sections in the order described above to determine which ones apply to the current request. When the first section matches, it is considered the current configuration for this request. If a subsequent section matches too, then each module with a directive in either of the sections is given a chance to merge its configuration between the two sections. The result is a third configuration, and the process goes on until all the configuration sections are evaluated.

After the above step, the "real" processing of the HTTP request begins: each module has a chance to run and perform whatever tasks they like. They can retrieve their own final merged configuration from the core of the `httpd` to determine how they should act.

An example can help to visualize the whole process. The following configuration uses the `Header` directive of `mod_headers` to set a specific HTTP header. What value will `httpd` set in the `CustomHeaderName` header for a request to `/example/index.html` ?

```
<Directory "/">
  Header set CustomHeaderName one
```

```
<FilesMatch ".*">
    Header set CustomHeaderName three
</FilesMatch>
</Directory>

<Directory "/example">
    Header set CustomHeaderName two
</Directory>
```

- **Directory "/"** matches and an initial configuration to set the CustomHeaderName header with the value one is created.
- **Directory "/example"** matches, and since [mod_headers](#) specifies in its code to override in case of a merge, a new configuration is created to set the CustomHeaderName header with the value two.
- **FilesMatch ".*"** matches and another merge opportunity arises, causing the CustomHeaderName header to be set with the value three.
- Eventually during the next steps of the HTTP request processing [mod_headers](#) will be called and it will receive the configuration to set the CustomHeaderName header with the value three. [mod_headers](#) normally uses this configuration to perform its job, namely setting the foo header. This does not mean that a module can't perform a more complex action like discarding directives because not needed or deprecated, etc..

This is true for .htaccess too since they have the same priority as **Directory** in the merge order. The important concept to understand is that configuration sections like **Directory** and **FilesMatch** are not comparable to module specific directives like [Header](#) or [RewriteRule](#) because they operate on different levels.

Some Examples

Below is an artificial example to show the order of merging. Assuming

they all apply to the request, the directives in this example will be applied in the order A > B > C > D > E.

```
<Location "/">
    E
</Location>

<Files "f.html">
    D
</Files>

<VirtualHost *>
<Directory "/a/b">
    B
</Directory>
</VirtualHost>

<DirectoryMatch "^.*b$">
    C
</DirectoryMatch>

<Directory "/a/b">
    A
</Directory>
```

For a more concrete example, consider the following. Regardless of any access restrictions placed in [<Directory>](#) sections, the [<Location>](#) section will be evaluated last and will allow unrestricted access to the server. In other words, order of merging is important, so be careful!

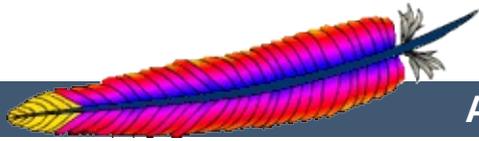
```
<Location "/">
    Require all granted
</Location>

# Woops! This <Directory> section will have no effect
<Directory "/">
```

```
<RequireAll>
    Require all granted
    Require not host badguy.example.com
</RequireAll>
</Directory>
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Caching Guide

This document supplements the [mod_cache](#), [mod_disk_cache](#), [mod_mem_cache](#), [mod_file_cache](#) and [htcacheclean](#) reference documentation. It describes how to use Apache's caching features to accelerate web and proxy serving, while avoiding common problems and misconfigurations.



As of Apache HTTP server version 2.2 [mod_cache](#) and [mod_file_cache](#) are no longer marked experimental and are considered suitable for production use. These caching architectures provide a powerful means to accelerate HTTP handling, both as an origin webserver and as a proxy.

[mod_cache](#) and its provider modules [mod_mem_cache](#) and [mod_disk_cache](#) provide intelligent, HTTP-aware caching. The content itself is stored in the cache, and `mod_cache` aims to honour all of the various HTTP headers and options that control the cachability of content. It can handle both local and proxied content. [mod_cache](#) is aimed at both simple and complex caching configurations, where you are dealing with proxied content, dynamic local content or have a need to speed up access to local files which change with time.

[mod_file_cache](#) on the other hand presents a more basic, but sometimes useful, form of caching. Rather than maintain the complexity of actively ensuring the cachability of URLs, [mod_file_cache](#) offers file-handle and memory-mapping tricks to keep a cache of files as they were when Apache was last started. As such, [mod_file_cache](#) is aimed at improving the access time to local static files which do not change very often.

As [mod_file_cache](#) presents a relatively simple caching implementation, apart from the specific sections on [CacheFile](#) and [MMapFile](#), the explanations in this guide cover the [mod_cache](#) caching architecture.

To get the most from this document, you should be familiar with the basics of HTTP, and have read the Users' Guides to [Mapping URLs to the Filesystem](#) and [Content negotiation](#).



Related Modules	Related Directives
mod_cache	CacheEnable
mod_mem_cache	CacheDisable
mod_disk_cache	CacheFile
mod_file_cache	MMapFile
	UseCanonicalName
	CacheNegotiatedDocs

There are two main stages in [mod_cache](#) that can occur in the lifetime of a request. First, [mod_cache](#) is a URL mapping module, which means that if a URL has been cached, and the cached version of that URL has not expired, the request will be served directly by [mod_cache](#).

This means that any other stages that might ordinarily happen in the process of serving a request -- for example being handled by [mod_proxy](#), or [mod_rewrite](#) -- won't happen. But then this is the point of caching content in the first place.

If the URL is not found within the cache, [mod_cache](#) will add a [filter](#) to the request handling. After Apache has located the content by the usual means, the filter will be run as the content is served. If the content is determined to be cacheable, the content will be saved to the cache for future serving.

If the URL is found within the cache, but also found to have expired, the filter is added anyway, but [mod_cache](#) will create a conditional request to the backend, to determine if the cached version is still current. If the cached version is still current, its meta-information will be updated and the request will be served from the cache. If the cached version is no longer current, the cached version will be deleted and the filter will save the updated content to the cache as it is served.

Improving Cache Hits

When caching locally generated content, ensuring that [UseCanonicalName](#) is set to On can dramatically improve the ratio of cache hits. This is because the hostname of the virtual-host serving the content forms a part of the cache key. With the setting set to On virtual-hosts with multiple server names or aliases will not produce differently cached entities, and instead content will be cached as per the canonical hostname.

Because caching is performed within the URL to filename translation phase, cached documents will only be served in response to URL requests. Ordinarily this is of little consequence, but there is one circumstance in which it matters: If you are using [Server Side Includes](#);

```
<!-- The following include can be cached -->
<!--#include virtual="/footer.html" -->

<!-- The following include can not be cached -->
<!--#include file="/path/to/footer.html" -->
```

If you are using Server Side Includes, and want the benefit of speedy serves from the cache, you should use `virtual` include types.

Expiry Periods

The default expiry period for cached entities is one hour, however this can be easily over-ridden by using the [CacheDefaultExpire](#) directive. This default is only used when the original source of the content does not specify an expire time or time of last modification.

If a response does not include an Expires header but does include a Last-Modified header, [mod_cache](#) can infer an expiry period based on the use of the [CacheLastModifiedFactor](#) directive.

For local content, [mod_expires](#) may be used to fine-tune the expiry period.

The maximum expiry period may also be controlled by using the [CacheMaxExpire](#).

A Brief Guide to Conditional Requests

When content expires from the cache and is re-requested from the backend or content provider, rather than pass on the original request, Apache will use a conditional request instead.

HTTP offers a number of headers which allow a client, or cache to discern between different versions of the same content. For example if a resource was served with an "Etag:" header, it is possible to make a conditional request with an "If-None-Match:" header. If a resource was served with a "Last-Modified:" header it is possible to make a conditional request with an "If-Modified-Since:" header, and so on.

When such a conditional request is made, the response differs depending on whether the content matches the conditions. If a request is made with an "If-Modified-Since:" header, and the content has not been modified since the time indicated in the request then a terse "304 Not Modified" response is issued.

If the content has changed, then it is served as if the request were not conditional to begin with.

The benefits of conditional requests in relation to caching are twofold. Firstly, when making such a request to the backend, if the content from the backend matches the content in the store, this can be determined easily and without the overhead of transferring the entire resource.

Secondly, conditional requests are usually less strenuous on the backend. For static files, typically all that is involved is a call to `stat()` or similar system call, to see if the file has changed in size or modification time. As such, even if Apache is caching local content, even expired content may still be served faster from the cache if it has not changed. As long as reading from the cache store is faster than reading from the

backend (e.g. an in-memory cache compared to reading from disk).

What Can be Cached?

As mentioned already, the two styles of caching in Apache work differently, [mod file cache](#) caching maintains file contents as they were when Apache was started. When a request is made for a file that is cached by this module, it is intercepted and the cached file is served.

[mod cache](#) caching on the other hand is more complex. When serving a request, if it has not been cached previously, the caching module will determine if the content is cacheable. The conditions for determining cachability of a response are;

1. Caching must be enabled for this URL. See the [CacheEnable](#) and [CacheDisable](#) directives.
2. The response must have a HTTP status code of 200, 203, 300, 301 or 410.
3. The request must be a HTTP GET request.
4. If the request contains an "Authorization:" header, the response will not be cached.
5. If the response contains an "Authorization:" header, it must also contain an "s-maxage", "must-revalidate" or "public" option in the "Cache-Control:" header.
6. If the URL included a query string (e.g. from a HTML form GET method) it will not be cached unless the response specifies an explicit expiration by including an "Expires:" header or the max-age or s-maxage directive of the "Cache-Control:" header, as per RFC2616 sections 13.9 and 13.2.1.
7. If the response has a status of 200 (OK), the response must also include at least one of the "Etag", "Last-Modified" or the "Expires" headers, or the max-age or s-maxage directive of the "Cache-

Control:" header, unless the [CacheIgnoreNoLastMod](#) directive has been used to require otherwise.

8. If the response includes the "private" option in a "Cache-Control:" header, it will not be stored unless the [CacheStorePrivate](#) has been used to require otherwise.
9. Likewise, if the response includes the "no-store" option in a "Cache-Control:" header, it will not be stored unless the [CacheStoreNoStore](#) has been used.
10. A response will not be stored if it includes a "Vary:" header containing the match-all "*".

What Should Not be Cached?

In short, any content which is highly time-sensitive, or which varies depending on the particulars of the request that are not covered by HTTP negotiation, should not be cached.

If you have dynamic content which changes depending on the IP address of the requester, or changes every 5 minutes, it should almost certainly not be cached.

If on the other hand, the content served differs depending on the values of various HTTP headers, it might be possible to cache it intelligently through the use of a "Vary" header.

Variable/Negotiated Content

If a response with a "Vary" header is received by [mod_cache](#) when requesting content by the backend it will attempt to handle it intelligently. If possible, [mod_cache](#) will detect the headers attributed in the "Vary" response in future requests and serve the correct cached response.

If for example, a response is received with a vary header such as;

Vary: negotiate, accept-language, accept-charset

mod_cache will only serve the cached content to requesters with accept-language and accept-charset headers matching those of the original request.



Authorization and Access Control

Using `mod_cache` is very much like having a built in reverse-proxy. Requests will be served by the caching module unless it determines that the backend should be queried. When caching local resources, this drastically changes the security model of Apache.

As traversing a filesystem hierarchy to examine potential `.htaccess` files would be a very expensive operation, partially defeating the point of caching (to speed up requests), `mod_cache` makes no decision about whether a cached entity is authorised for serving. In other words; if `mod_cache` has cached some content, it will be served from the cache as long as that content has not expired.

If, for example, your configuration permits access to a resource by IP address you should ensure that this content is not cached. You can do this by using the `CacheDisable` directive, or `mod_expires`. Left unchecked, `mod_cache` - very much like a reverse proxy - would cache the content when served and then serve it to any client, on any IP address.

Local exploits

As requests to end-users can be served from the cache, the cache itself can become a target for those wishing to deface or interfere with content. It is important to bear in mind that the cache must at all times be writable by the user which Apache is running as. This is in stark contrast to the usually recommended situation of maintaining all content unwritable by the Apache user.

If the Apache user is compromised, for example through a flaw in a CGI process, it is possible that the cache may be targeted. When using `mod_disk_cache`, it is relatively easy to insert or modify a cached entity.

This presents a somewhat elevated risk in comparison to the other types of attack it is possible to make as the Apache user. If you are using [mod_disk_cache](#) you should bear this in mind - ensure you upgrade Apache when security upgrades are announced and run CGI processes as a non-Apache user using [suEXEC](#) if possible.

Cache Poisoning

When running Apache as a caching proxy server, there is also the potential for so-called cache poisoning. Cache Poisoning is a broad term for attacks in which an attacker causes the proxy server to retrieve incorrect (and usually undesirable) content from the backend.

For example if the DNS servers used by your system running Apache are vulnerable to DNS cache poisoning, an attacker may be able to control where Apache connects to when requesting content from the origin server. Another example is so-called HTTP request-smuggling attacks.

This document is not the correct place for an in-depth discussion of HTTP request smuggling (instead, try your favourite search engine) however it is important to be aware that it is possible to make a series of requests, and to exploit a vulnerability on an origin webserver such that the attacker can entirely control the content retrieved by the proxy.



Related Modules	Related Directives
mod_file_cache	CacheFile
mod_mem_cache	CacheEnable
	CacheDisable

The act of opening a file can itself be a source of delay, particularly on network filesystems. By maintaining a cache of open file descriptors for commonly served files, Apache can avoid this delay. Currently Apache provides two different implementations of File-Handle Caching.

CacheFile

The most basic form of caching present in Apache is the file-handle caching provided by [mod_file_cache](#). Rather than caching file-contents, this cache maintains a table of open file descriptors. Files to be cached in this manner are specified in the configuration file using the [CacheFile](#) directive.

The [CacheFile](#) directive instructs Apache to open the file when Apache is started and to re-use this file-handle for all subsequent access to this file.

```
CacheFile /usr/local/apache2/htdocs/index.html
```

If you intend to cache a large number of files in this manner, you must ensure that your operating system's limit for the number of open files is set appropriately.

Although using [CacheFile](#) does not cause the file-contents to be cached per-se, it does mean that if the file changes while Apache is running these changes will not be picked up. The file will be consistently served as it was when Apache was started.

If the file is removed while Apache is running, Apache will continue to maintain an open file descriptor and serve the file as it was when Apache was started. This usually also means that although the file will have been deleted, and not show up on the filesystem, extra free space will not be recovered until Apache is stopped and the file descriptor closed.

CacheEnable fd

[mod_mem_cache](#) also provides its own file-handle caching scheme, which can be enabled via the [CacheEnable](#) directive.

```
CacheEnable fd /
```

As with all of [mod_cache](#) this type of file-handle caching is intelligent, and handles will not be maintained beyond the expiry time of the cached content.



Related Modules	Related Directives
mod_mem_cache	CacheEnable
mod_file_cache	CacheDisable
	MMapFile

Serving directly from system memory is universally the fastest method of serving content. Reading files from a disk controller or, even worse, from a remote network is orders of magnitude slower. Disk controllers usually involve physical processes, and network access is limited by your available bandwidth. Memory access on the other hand can take mere nano-seconds.

System memory isn't cheap though, byte for byte it's by far the most expensive type of storage and it's important to ensure that it is used efficiently. By caching files in memory you decrease the amount of memory available on the system. As we'll see, in the case of operating system caching, this is not so much of an issue, but when using Apache's own in-memory caching it is important to make sure that you do not allocate too much memory to a cache. Otherwise the system will be forced to swap out memory, which will likely degrade performance.

Operating System Caching

Almost all modern operating systems cache file-data in memory managed directly by the kernel. This is a powerful feature, and for the most part operating systems get it right. For example, on Linux, let's look at the difference in the time it takes to read a file for the first time and the second time;

```
colm@coroebus:~$ time cat testfile > /dev/null
real    0m0.065s
user    0m0.000s
sys     0m0.001s
colm@coroebus:~$ time cat testfile > /dev/null
real    0m0.003s
```

```
user    0m0.003s
sys     0m0.000s
```

Even for this small file, there is a huge difference in the amount of time it takes to read the file. This is because the kernel has cached the file contents in memory.

By ensuring there is "spare" memory on your system, you can ensure that more and more file-contents will be stored in this cache. This can be a very efficient means of in-memory caching, and involves no extra configuration of Apache at all.

Additionally, because the operating system knows when files are deleted or modified, it can automatically remove file contents from the cache when necessary. This is a big advantage over Apache's in-memory caching which has no way of knowing when a file has changed.

Despite the performance and advantages of automatic operating system caching there are some circumstances in which in-memory caching may be better performed by Apache.

Firstly, an operating system can only cache files it knows about. If you are running Apache as a proxy server, the files you are caching are not locally stored but remotely served. If you still want the unbeatable speed of in-memory caching, Apache's own memory caching is needed.

MMapFile Caching

`mod_file_cache` provides the `MMapFile` directive, which allows you to have Apache map a static file's contents into memory at start time (using the `mmap` system call). Apache will use the in-memory contents for all subsequent accesses to this file.

```
MMapFile /usr/local/apache2/htdocs/index.html
```

As with the `CacheFile` directive, any changes in these files will not be

picked up by Apache after it has started.

The [MMapFile](#) directive does not keep track of how much memory it allocates, so you must ensure not to over-use the directive. Each Apache child process will replicate this memory, so it is critically important to ensure that the files mapped are not so large as to cause the system to swap memory.

mod_mem_cache Caching

[mod_mem_cache](#) provides a HTTP-aware intelligent in-memory cache. It also uses heap memory directly, which means that even if *MMap* is not supported on your system, [mod_mem_cache](#) may still be able to perform caching.

Caching of this type is enabled via;

```
# Enable memory caching
CacheEnable mem /

# Limit the size of the cache to 1 Megabyte
MCacheSize 1024
```



Related Modules	Related Directives
mod_disk_cache	CacheEnable CacheDisable

[mod_disk_cache](#) provides a disk-based caching mechanism for [mod_cache](#). As with [mod_mem_cache](#) this cache is intelligent and content will be served from the cache only as long as it is considered valid.

Typically the module will be configured as so;

```
CacheRoot    /var/cache/apache/  
CacheEnable  disk /  
CacheDirLevels 2  
CacheDirLength 1
```

Importantly, as the cached files are locally stored, operating system in-memory caching will typically be applied to their access also. So although the files are stored on disk, if they are frequently accessed it is likely the operating system will ensure that they are actually served from memory.

Understanding the Cache-Store

To store items in the cache, [mod_disk_cache](#) creates a 22 character hash of the URL being requested. This hash incorporates the hostname, protocol, port, path and any CGI arguments to the URL, to ensure that multiple URLs do not collide.

Each character may be any one of 64-different characters, which mean that overall there are 64^{22} possible hashes. For example, a URL might be hashed to xyTGxSM02b68mBCykqkp1w. This hash is used as a prefix for the naming of the files specific to that URL within the cache, however first it is split up into directories as per the [CacheDirLevels](#) and [CacheDirLength](#) directives.

CacheDirLevels specifies how many levels of subdirectory there should be, and CacheDirLength specifies how many characters should be in each directory. With the example settings given above, the hash would be turned into a filename prefix as `/var/cache/apache/x/y/TGxSM02b68mBCykqkp1w`.

The overall aim of this technique is to reduce the number of subdirectories or files that may be in a particular directory, as most file-systems slow down as this number increases. With setting of "1" for CacheDirLength there can at most be 64 subdirectories at any particular level. With a setting of 2 there can be $64 * 64$ subdirectories, and so on. Unless you have a good reason not to, using a setting of "1" for CacheDirLength is recommended.

Setting CacheDirLevels depends on how many files you anticipate to store in the cache. With the setting of "2" used in the above example, a grand total of 4096 subdirectories can ultimately be created. With 1 million files cached, this works out at roughly 245 cached URLs per directory.

Each URL uses at least two files in the cache-store. Typically there is a ".header" file, which includes meta-information about the URL, such as when it is due to expire and a ".data" file which is a verbatim copy of the content to be served.

In the case of a content negotiated via the "Vary" header, a ".vary" directory will be created for the URL in question. This directory will have multiple ".data" files corresponding to the differently negotiated content.

Maintaining the Disk Cache

Although mod_disk_cache will remove cached content as it is expired, it does not maintain any information on the total size of the cache or how little free space may be left.

Instead, provided with Apache is the [htcacheclean](#) tool which, as the name suggests, allows you to clean the cache periodically. Determining how frequently to run [htcacheclean](#) and what target size to use for the cache is somewhat complex and trial and error may be needed to select optimal values.

[htcacheclean](#) has two modes of operation. It can be run as persistent daemon, or periodically from cron. [htcacheclean](#) can take up to an hour or more to process very large (tens of gigabytes) caches and if you are running it from cron it is recommended that you determine how long a typical run takes, to avoid running more than one instance at a time.

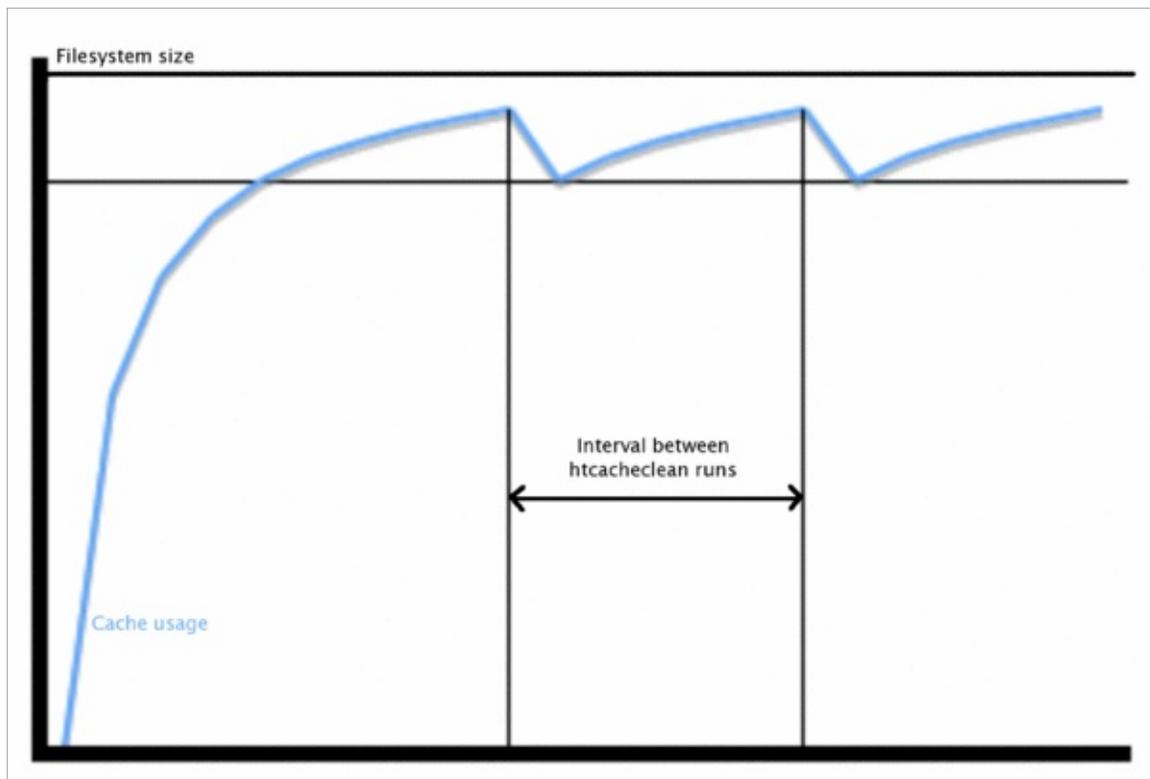
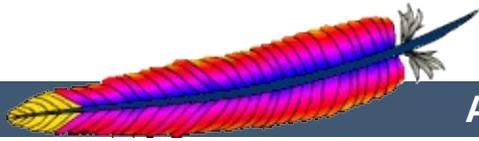


Figure 1: Typical cache growth / clean sequence.

Because [mod_disk_cache](#) does not itself pay attention to how much space is used you should ensure that [htcacheclean](#) is configured to leave enough "grow room" following a clean.

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Server-Wide Configuration

This document explains some of the directives provided by the [core](#) server which are used to configure the basic operations of the server.



Related Modules	Related Directives
	<u>ServerName</u>
	<u>ServerAdmin</u>
	<u>ServerSignature</u>
	<u>ServerTokens</u>
	<u>UseCanonicalName</u>
	<u>UseCanonicalPhysicalPort</u>

The [ServerAdmin](#) and [ServerTokens](#) directives control what information about the server will be presented in server-generated documents such as error messages. The [ServerTokens](#) directive sets the value of the Server HTTP response header field.

The [ServerName](#), [UseCanonicalName](#) and [UseCanonicalPhysicalPort](#) directives are used by the server to determine how to construct self-referential URLs. For example, when a client requests a directory, but does not include the trailing slash in the directory name, Apache must redirect the client to the full name including the trailing slash so that the client will correctly resolve relative references in the document.



Related Modules	Related Directives
-----------------	--------------------

	CoreDumpDirectory
--	-----------------------------------

	DocumentRoot
--	------------------------------

	ErrorLog
--	--------------------------

	LockFile
--	--------------------------

	PidFile
--	-------------------------

	ScoreBoardFile
--	--------------------------------

	ServerRoot
--	----------------------------

These directives control the locations of the various files that Apache needs for proper operation. When the pathname used does not begin with a slash (/), the files are located relative to the [ServerRoot](#). Be careful about locating files in paths which are writable by non-root users. See the [security tips](#) documentation for more details.

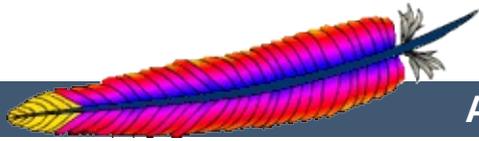


Related Modules	Related Directives
	LimitRequestBody
	LimitRequestFields
	LimitRequestFieldsize
	LimitRequestLine
	RLimitCPU
	RLimitMEM
	RLimitNPROC
	ThreadStackSize

The [LimitRequest*](#) directives are used to place limits on the amount of resources Apache will use in reading requests from clients. By limiting these values, some kinds of denial of service attacks can be mitigated.

The [RLimit*](#) directives are used to limit the amount of resources which can be used by processes forked off from the Apache children. In particular, this will control resources used by CGI scripts and SSI exec commands.

The [ThreadStackSize](#) directive is used with some platforms to control the stack size.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Log Files

In order to effectively manage a web server, it is necessary to get feedback about the activity and performance of the server as well as any problems that may be occurring. The Apache HTTP Server provides very comprehensive and flexible logging capabilities. This document describes how to configure its logging capabilities, and how to understand what the logs contain.



Anyone who can write to the directory where Apache is writing a log file can almost certainly gain access to the uid that the server is started as, which is normally root. Do *NOT* give people write access to the directory the logs are stored in without being aware of the consequences; see the [security tips](#) document for details.

In addition, log files may contain information supplied directly by the client, without escaping. Therefore, it is possible for malicious clients to insert control-characters in the log files, so care must be taken in dealing with raw logs.



Related Modules

Related Directives

[ErrorLog](#)

[LogLevel](#)

The server error log, whose name and location is set by the [ErrorLog](#) directive, is the most important log file. This is the place where Apache httpd will send diagnostic information and record any errors that it encounters in processing requests. It is the first place to look when a problem occurs with starting the server or with the operation of the server, since it will often contain details of what went wrong and how to fix it.

The error log is usually written to a file (typically `error_log` on Unix systems and `error.log` on Windows and OS/2). On Unix systems it is also possible to have the server send errors to `syslog` or [pipe them to a program](#).

The format of the error log is relatively free-form and descriptive. But there is certain information that is contained in most error log entries. For example, here is a typical message.

```
[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1] client denied by server configuration: /export/home/live/ap/htdocs/test
```

The first item in the log entry is the date and time of the message. The second item lists the severity of the error being reported. The [LogLevel](#) directive is used to control the types of errors that are sent to the error log by restricting the severity level. The third item gives the IP address of the client that generated the error. Beyond that is the message itself, which in this case indicates that the server has been configured to deny the client access. The server reports the file-system path (as opposed to the web path) of the requested document.

A very wide variety of different messages can appear in the error log. Most look similar to the example above. The error log will also contain debugging output from CGI scripts. Any information written to `stderr` by a CGI script will be copied directly to the error log.

It is not possible to customize the error log by adding or removing information. However, error log entries dealing with particular requests have corresponding entries in the [access log](#). For example, the above example entry corresponds to an access log entry with status code 403. Since it is possible to customize the access log, you can obtain more information about error conditions using that log file.

During testing, it is often useful to continuously monitor the error log for any problems. On Unix systems, you can accomplish this using:

```
tail -f error_log
```



Related Modules	Related Directives
mod_log_config	CustomLog
mod_setenvif	LogFormat
	SetEnvIf

The server access log records all requests processed by the server. The location and content of the access log are controlled by the [CustomLog](#) directive. The [LogFormat](#) directive can be used to simplify the selection of the contents of the logs. This section describes how to configure the server to record information in the access log.

Of course, storing the information in the access log is only the start of log management. The next step is to analyze this information to produce useful statistics. Log analysis in general is beyond the scope of this document, and not really part of the job of the web server itself. For more information about this topic, and for applications which perform log analysis, check the [Open Directory](#) or [Yahoo](#).

Various versions of Apache httpd have used other modules and directives to control access logging, including `mod_log_referer`, `mod_log_agent`, and the `TransferLog` directive. The [CustomLog](#) directive now subsumes the functionality of all the older directives.

The format of the access log is highly configurable. The format is specified using a format string that looks much like a C-style `printf(1)` format string. Some examples are presented in the next sections. For a complete list of the possible contents of the format string, see the [mod_log_config format strings](#).

Common Log Format

A typical configuration for the access log might look as follows.

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

This defines the *nickname* common and associates it with a particular log format string. The format string consists of percent directives, each of which tell the server to log a particular piece of information. Literal characters may also be placed in the format string and will be copied directly into the log output. The quote character (") must be escaped by placing a backslash before it to prevent it from being interpreted as the end of the format string. The format string may also contain the special control characters "\n" for new-line and "\t" for tab.

The [CustomLog](#) directive sets up a new log file using the defined *nickname*. The filename for the access log is relative to the [ServerRoot](#) unless it begins with a slash.

The above configuration will write log entries in a format known as the Common Log Format (CLF). This standard format can be produced by many different web servers and read by many log analysis programs. The log file entries produced in CLF will look something like this:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif
HTTP/1.0" 200 2326
```

Each part of this log entry is described below.

127.0.0.1 (%h)

This is the IP address of the client (remote host) which made the request to the server. If [HostnameLookups](#) is set to On, then the server will try to determine the hostname and log it in place of the IP address. However, this configuration is not recommended since it can significantly slow the server. Instead, it is best to use a log post-processor such as [logresolve](#) to determine the hostnames. The IP address reported here is not necessarily the address of the machine at which the user is sitting. If a proxy server exists between

the user and the server, this address will be the address of the proxy, rather than the originating machine.

- (%l)

The "hyphen" in the output indicates that the requested piece of information is not available. In this case, the information that is not available is the RFC 1413 identity of the client determined by `identd` on the clients machine. This information is highly unreliable and should almost never be used except on tightly controlled internal networks. Apache `httpd` will not even attempt to determine this information unless `IdentityCheck` is set to `On`.

frank (%u)

This is the userid of the person requesting the document as determined by HTTP authentication. The same value is typically provided to CGI scripts in the `REMOTE_USER` environment variable. If the status code for the request (see below) is 401, then this value should not be trusted because the user is not yet authenticated. If the document is not password protected, this part will be "-" just like the previous one.

[10/Oct/2000:13:55:36 -0700] (%t)

The time that the request was received. The format is:

```
[day/month/year:hour:minute:second zone]
day = 2*digit
month = 3*letter
year = 4*digit
hour = 2*digit
minute = 2*digit
second = 2*digit
zone = ('+' | '-' ) 4*digit
```

It is possible to have the time displayed in another format by specifying `%{format}t` in the log format string, where `format` is as in `strftime(3)` from the C standard library.

"GET /apache_pb.gif HTTP/1.0" (\%r\)

The request line from the client is given in double quotes. The request line contains a great deal of useful information. First, the method used by the client is GET. Second, the client requested the resource /apache_pb.gif, and third, the client used the protocol HTTP/1.0. It is also possible to log one or more parts of the request line independently. For example, the format string "%m %U%q %H" will log the method, path, query-string, and protocol, resulting in exactly the same output as "%r".

200 (%>s)

This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2), a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5). The full list of possible status codes can be found in the [HTTP specification](#) (RFC2616 section 10).

2326 (%b)

The last part indicates the size of the object returned to the client, not including the response headers. If no content was returned to the client, this value will be "-". To log "0" for no content, use %B instead.

Combined Log Format

Another commonly used format string is called the Combined Log Format. It can be used as follows.

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined
CustomLog log/access_log combined
```

This format is exactly the same as the Common Log Format, with the addition of two more fields. Each of the additional fields uses the percent-directive `%{header}i`, where *header* can be any HTTP request header.

The access log under this format will look like:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif
HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08
[en] (Win98; I ;Nav)"
```

The additional fields are:

"http://www.example.com/start.html" (\%{Referer}i\)"

The "Referer" (sic) HTTP request header. This gives the site that the client reports having been referred from. (This should be the page that links to or includes /apache_pb.gif).

"Mozilla/4.08 [en] (Win98; I ;Nav)" (\%{User-agent}i\)"

The User-Agent HTTP request header. This is the identifying information that the client browser reports about itself.

Multiple Access Logs

Multiple access logs can be created simply by specifying multiple [CustomLog](#) directives in the configuration file. For example, the following directives will create three access logs. The first contains the basic CLF information, while the second and third contain referer and browser information. The last two [CustomLog](#) lines show how to mimic the effects of the ReferLog and AgentLog directives.

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
CustomLog logs/referer_log "%{Referer}i -> %U"
CustomLog logs/agent_log "%{User-agent}i"
```

This example also shows that it is not necessary to define a nickname with the [LogFormat](#) directive. Instead, the log format can be specified directly in the [CustomLog](#) directive.

Conditional Logs

There are times when it is convenient to exclude certain entries from the access logs based on characteristics of the client request. This is easily accomplished with the help of [environment variables](#). First, an environment variable must be set to indicate that the request meets certain conditions. This is usually accomplished with `SetEnvIf`. Then the `env=` clause of the `CustomLog` directive is used to include or exclude requests where the environment variable is set. Some examples:

```
# Mark requests from the loop-back interface
SetEnvIf Remote_Addr "127\.0\.0\.1" dontlog
# Mark requests for the robots.txt file
SetEnvIf Request_URI "^/robots\.txt$" dontlog
# Log what remains
CustomLog logs/access_log common env=!dontlog
```

As another example, consider logging requests from english-speakers to one log file, and non-english speakers to a different log file.

```
SetEnvIf Accept-Language "en" english
CustomLog logs/english_log common env=english
CustomLog logs/non_english_log common env=!english
```

Although we have just shown that conditional logging is very powerful and flexible, it is not the only way to control the contents of the logs. Log files are more useful when they contain a complete record of server activity. It is often easier to simply post-process the log files to remove requests that you do not want to consider.



Log Rotation

On even a moderately busy server, the quantity of information stored in the log files is very large. The access log file typically grows 1 MB or more per 10,000 requests. It will consequently be necessary to periodically rotate the log files by moving or deleting the existing logs. This cannot be done while the server is running, because Apache will continue writing to the old log file as long as it holds the file open. Instead, the server must be [restarted](#) after the log files are moved or deleted so that it will open new log files.

By using a *graceful* restart, the server can be instructed to open new log files without losing any existing or pending connections from clients. However, in order to accomplish this, the server must continue to write to the old log files while it finishes serving old requests. It is therefore necessary to wait for some time after the restart before doing any processing on the log files. A typical scenario that simply rotates the logs and compresses the old logs to save space is:

```
mv access_log access_log.old
mv error_log error_log.old
apachectl graceful
sleep 600
gzip access_log.old error_log.old
```

Another way to perform log rotation is using [piped logs](#) as discussed in the next section.



Piped Logs

Apache httpd is capable of writing error and access log files through a pipe to another process, rather than directly to a file. This capability dramatically increases the flexibility of logging, without adding code to the main server. In order to write logs to a pipe, simply replace the filename with the pipe character "|", followed by the name of the executable which should accept log entries on its standard input. Apache will start the piped-log process when the server starts, and will restart it if it crashes while the server is running. (This last feature is why we can refer to this technique as "reliable piped logging".)

Piped log processes are spawned by the parent Apache httpd process, and inherit the userid of that process. This means that piped log programs usually run as root. It is therefore very important to keep the programs simple and secure.

One important use of piped logs is to allow log rotation without having to restart the server. The Apache HTTP Server includes a simple program called [rotatelogs](#) for this purpose. For example, to rotate the logs every 24 hours, you can use:

```
CustomLog "|/usr/local/apache/bin/rotatelogs /var/log/access_log  
86400" common
```

Notice that quotes are used to enclose the entire command that will be called for the pipe. Although these examples are for the access log, the same technique can be used for the error log.

As with conditional logging, piped logs are a very powerful tool, but they should not be used where a simpler solution like off-line post-processing is available.

By default the piped log process is spawned using a shell. (usually with `/bin/sh -c`). Depending on the shell specifics invocation via shell might lead to an additional shell process for the lifetime of the logging

pipe program and signal handling problems during restart.

Use "`||`" instead of "`|`" to spawn without invoking a shell:

```
# Invoke "rotatelogs" without using a shell
CustomLog "||/usr/local/apache/bin/rotatelogs /var/log/access_log
86400" common
```

Windows note

Note that on Windows, you may run into problems when running many piped logger processes, especially when HTTPD is running as a service. This is caused by running out of desktop heap space. The desktop heap space given to each service is specified by the third argument to the `SharedSection` parameter in the `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SessionMa` registry value. **Change this value with care**; the normal caveats for changing the Windows registry apply, but you might also exhaust the desktop heap pool if the number is adjusted too high.



When running a server with many [virtual hosts](#), there are several options for dealing with log files. First, it is possible to use logs exactly as in a single-host server. Simply by placing the logging directives outside the `<VirtualHost>` sections in the main server context, it is possible to log all requests in the same access log and error log. This technique does not allow for easy collection of statistics on individual virtual hosts.

If `CustomLog` or `ErrorLog` directives are placed inside a `<VirtualHost>` section, all requests or errors for that virtual host will be logged only to the specified file. Any virtual host which does not have logging directives will still have its requests sent to the main server logs. This technique is very useful for a small number of virtual hosts, but if the number of hosts is very large, it can be complicated to manage. In addition, it can often create problems with [insufficient file descriptors](#).

For the access log, there is a very good compromise. By adding information on the virtual host to the log format string, it is possible to log all hosts to the same log, and later split the log into individual files. For example, consider the following directives.

```
LogFormat "%v %l %u %t \"%r\" %>s %b" comonvhost
CustomLog logs/access_log comonvhost
```

The `%v` is used to log the name of the virtual host that is serving the request. Then a program like [split-logfile](#) can be used to post-process the access log in order to split it into one file per virtual host.



Related Modules	Related Directives
mod_logio	LogFormat
mod_log_forensic	BufferedLogs
mod_cgi	ForensicLog
mod_rewrite	PidFile
mod_log_config	RewriteLog
	RewriteLogLevel
	ScriptLog
	ScriptLogBuffer
	ScriptLogLength

Logging actual bytes sent and received

[mod_logio](#) adds in two additional [LogFormat](#) fields (%I and %O) that log the actual number of bytes received and sent on the network.

Forensic Logging

[mod_log_forensic](#) provides for forensic logging of client requests. Logging is done before and after processing a request, so the forensic log contains two log lines for each request. The forensic logger is very strict with no customizations. It can be an invaluable debugging and security tool.

PID File

On startup, Apache httpd saves the process id of the parent httpd process to the file `logs/httpd.pid`. This filename can be changed with the [PidFile](#) directive. The process-id is for use by the administrator in restarting and terminating the daemon by sending signals to the parent process; on Windows, use the `-k` command line option instead. For more information see the [Stopping and Restarting](#) page.

Script Log

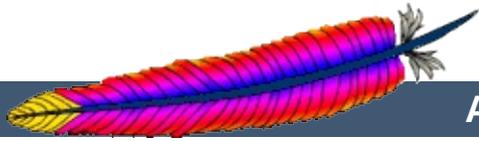
In order to aid in debugging, the [ScriptLog](#) directive allows you to record the input to and output from CGI scripts. This should only be used in testing - not for live servers. More information is available in the [mod_cgi](#) documentation.

Rewrite Log

When using the powerful and complex features of [mod_rewrite](#), it is almost always necessary to use the [RewriteLog](#) to help in debugging. This log file produces a detailed analysis of how the rewriting engine transforms requests. The level of detail is controlled by the [RewriteLogLevel](#) directive.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Mapping URLs to Filesystem Locations

This document explains how Apache uses the URL of a request to determine the filesystem location from which to serve a file.



Related Modules	Related Directives
mod_alias	Alias
mod_proxy	AliasMatch
mod_rewrite	CheckSpelling
mod_userdir	DocumentRoot
mod_speling	ErrorDocument
mod_vhost_alias	Options
	ProxyPass
	ProxyPassReverse
	ProxyPassReverseCookieDomain
	ProxyPassReverseCookiePath
	Redirect
	RedirectMatch
	RewriteCond
	RewriteRule
	ScriptAlias
	ScriptAliasMatch
	UserDir



In deciding what file to serve for a given request, Apache's default behavior is to take the URL-Path for the request (the part of the URL following the hostname and port) and add it to the end of the [DocumentRoot](#) specified in your configuration files. Therefore, the files and directories underneath the [DocumentRoot](#) make up the basic document tree which will be visible from the web.

For example, if [DocumentRoot](#) were set to `/var/www/html` then a request for `http://www.example.com/fish/guppies.html` would result in the file `/var/www/html/fish/guppies.html` being served to the requesting client.

Apache is also capable of [Virtual Hosting](#), where the server receives requests for more than one host. In this case, a different [DocumentRoot](#) can be specified for each virtual host, or alternatively, the directives provided by the module [mod_vhost_alias](#) can be used to dynamically determine the appropriate place from which to serve content based on the requested IP address or hostname.

The [DocumentRoot](#) directive is set in your main server configuration file (`httpd.conf`) and, possibly, once per additional [Virtual Host](#) you create.



There are frequently circumstances where it is necessary to allow web access to parts of the filesystem that are not strictly underneath the [DocumentRoot](#). Apache offers several different ways to accomplish this. On Unix systems, symbolic links can bring other parts of the filesystem under the [DocumentRoot](#). For security reasons, Apache will follow symbolic links only if the [Options](#) setting for the relevant directory includes `FollowSymLinks` or `SymLinksIfOwnerMatch`.

Alternatively, the [Alias](#) directive will map any part of the filesystem into the web space. For example, with

```
Alias /docs /var/web
```

the URL `http://www.example.com/docs/dir/file.html` will be served from `/var/web/dir/file.html`. The [ScriptAlias](#) directive works the same way, with the additional effect that all content located at the target path is treated as [CGI](#) scripts.

For situations where you require additional flexibility, you can use the [AliasMatch](#) and [ScriptAliasMatch](#) directives to do powerful [regular expression](#) based matching and substitution. For example,

```
ScriptAliasMatch ^/~([a-zA-Z0-9]+)/cgi-bin/(.+) /home/$1/cgi-bin/$2
```

will map a request to `http://example.com/~user/cgi-bin/script.cgi` to the path `/home/user/cgi-bin/script.cgi` and will treat the resulting file as a CGI script.



Traditionally on Unix systems, the home directory of a particular *user* can be referred to as `~user/`. The module `mod_userdir` extends this idea to the web by allowing files under each user's home directory to be accessed using URLs such as the following.

```
http://www.example.com/~user/file.html
```

For security reasons, it is inappropriate to give direct access to a user's home directory from the web. Therefore, the `UserDir` directive specifies a directory underneath the user's home directory where web files are located. Using the default setting of `UserDir public_html`, the above URL maps to a file at a directory like `/home/user/public_html/file.html` where `/home/user/` is the user's home directory as specified in `/etc/passwd`.

There are also several other forms of the `UserDir` directive which you can use on systems where `/etc/passwd` does not contain the location of the home directory.

Some people find the "~" symbol (which is often encoded on the web as `%7e`) to be awkward and prefer to use an alternate string to represent user directories. This functionality is not supported by `mod_userdir`. However, if users' home directories are structured in a regular way, then it is possible to use the `AliasMatch` directive to achieve the desired effect. For example, to make `http://www.example.com/upages/user/file.html` map to `/home/user/public_html/file.html`, use the following `AliasMatch` directive:

```
AliasMatch ^/upages/([a-zA-Z0-9]+)/?(.*) /home/$1/public_html/$2
```



URL REDIRECTION

The configuration directives discussed in the above sections tell Apache to get content from a specific place in the filesystem and return it to the client. Sometimes, it is desirable instead to inform the client that the requested content is located at a different URL, and instruct the client to make a new request with the new URL. This is called *redirection* and is implemented by the `Redirect` directive. For example, if the contents of the directory `/foo/` under the `DocumentRoot` are moved to the new directory `/bar/`, you can instruct clients to request the content at the new location as follows:

```
Redirect permanent /foo/ http://www.example.com/bar/
```

This will redirect any URL-Path starting in `/foo/` to the same URL path on the `www.example.com` server with `/bar/` substituted for `/foo/`. You can redirect clients to any server, not only the origin server.

Apache also provides a `RedirectMatch` directive for more complicated rewriting problems. For example, to redirect requests for the site home page to a different site, but leave all other requests alone, use the following configuration:

```
RedirectMatch permanent ^/$ http://www.example.com/startpage.html
```

Alternatively, to temporarily redirect all pages on one site to a particular page on another site, use the following:

```
RedirectMatch temp .* http://othersite.example.com/startpage.html
```



Reverse Proxy

Apache also allows you to bring remote documents into the URL space of the local server. This technique is called *reverse proxying* because the web server acts like a proxy server by fetching the documents from a remote server and returning them to the client. It is different from normal proxying because, to the client, it appears the documents originate at the reverse proxy server.

In the following example, when clients request documents under the `/foo/` directory, the server fetches those documents from the `/bar/` directory on `internal.example.com` and returns them to the client as if they were from the local server.

```
ProxyPass /foo/ http://internal.example.com/bar/  
ProxyPassReverse /foo/ http://internal.example.com/bar/  
ProxyPassReverseCookieDomain internal.example.com public.example.com  
ProxyPassReverseCookiePath /foo/ /bar/
```

The [ProxyPass](#) configures the server to fetch the appropriate documents, while the [ProxyPassReverse](#) directive rewrites redirects originating at `internal.example.com` so that they target the appropriate directory on the local server. Similarly, the [ProxyPassReverseCookieDomain](#) and [ProxyPassReverseCookiePath](#) rewrite cookies set by the backend server.

It is important to note, however, that links inside the documents will not be rewritten. So any absolute links on `internal.example.com` will result in the client breaking out of the proxy server and requesting directly from `internal.example.com`. A third-party module [mod_proxy_html](#) is available to rewrite links in HTML and XHTML.



Rewriting Engine

When even more powerful substitution is required, the rewriting engine provided by [mod_rewrite](#) can be useful. The directives provided by this module use characteristics of the request such as browser type or source IP address in deciding from where to serve content. In addition, `mod_rewrite` can use external database files or programs to determine how to handle a request. The rewriting engine is capable of performing all three types of mappings discussed above: internal redirects (aliases), external redirects, and proxying. Many practical examples employing `mod_rewrite` are discussed in the [detailed mod_rewrite documentation](#).



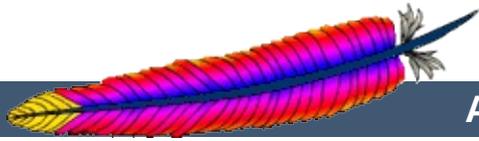
File Not Found

Inevitably, URLs will be requested for which no matching file can be found in the filesystem. This can happen for several reasons. In some cases, it can be a result of moving documents from one location to another. In this case, it is best to use [URL redirection](#) to inform clients of the new location of the resource. In this way, you can assure that old bookmarks and links will continue to work, even though the resource is at a new location.

Another common cause of "File Not Found" errors is accidental mistyping of URLs, either directly in the browser, or in HTML links. Apache provides the module [mod_spelling](#) (sic) to help with this problem. When this module is activated, it will intercept "File Not Found" errors and look for a resource with a similar filename. If one such file is found, `mod_spelling` will send an HTTP redirect to the client informing it of the correct location. If several "close" files are found, a list of available alternatives will be presented to the client.

An especially useful feature of `mod_spelling`, is that it will compare filenames without respect to case. This can help systems where users are unaware of the case-sensitive nature of URLs and the unix filesystem. But using `mod_spelling` for anything more than the occasional URL correction can place additional load on the server, since each "incorrect" request is followed by a URL redirection and a new request from the client.

If all attempts to locate the content fail, Apache returns an error page with HTTP status code 404 (file not found). The appearance of this page is controlled with the [ErrorDocument](#) directive and can be customized in a flexible manner as discussed in the [Custom error responses](#) document.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Dynamic Shared Object (DSO) Support

The Apache HTTP Server is a modular program where the administrator can choose the functionality to include in the server by selecting a set of modules. The modules can be statically compiled into the [httpd](#) binary when the server is built. Alternatively, modules can be compiled as Dynamic Shared Objects (DSOs) that exist separately from the main [httpd](#) binary file. DSO modules may be compiled at the time the server is built, or they may be compiled and added at a later time using the Apache Extension Tool ([apxs](#)).

This document describes how to use DSO modules as well as the theory behind their use.



Implementation

[mod_so](#)

[LoadModule](#)

The DSO support for loading individual Apache modules is based on a module named [mod_so](#) which must be statically compiled into the Apache core. It is the only module besides [core](#) which cannot be put into a DSO itself. Practically all other distributed Apache modules can then be placed into a DSO by individually enabling the DSO build for them via [configure](#)'s `--enable-module=shared` option as discussed in the [install documentation](#). After a module is compiled into a DSO named `mod_foo.so` you can use [mod_so](#)'s [LoadModule](#) command in your `httpd.conf` file to load this module at server startup or restart.

To simplify this creation of DSO files for Apache modules (especially for third-party modules) a new support program named [apxs](#) (*APache eXtenSion*) is available. It can be used to build DSO based modules *outside of* the Apache source tree. The idea is simple: When installing Apache the [configure](#)'s `make install` procedure installs the Apache C header files and puts the platform-dependent compiler and linker flags for building DSO files into the [apxs](#) program. This way the user can use [apxs](#) to compile his Apache module sources without the Apache distribution source tree and without having to fiddle with the platform-dependent compiler and linker flags for DSO support.



To give you an overview of the DSO features of Apache 2.x, here is a short and concise summary:

1. Build and install a *distributed* Apache module, say `mod_foo.c`, into its own DSO `mod_foo.so`:

```
$ ./configure --prefix=/path/to/install --enable-foo=shared
$ make install
```

2. Build and install a *third-party* Apache module, say `mod_foo.c`, into its own DSO `mod_foo.so`:

```
$ ./configure --add-
module=module_type:/path/to/3rdparty/mod_foo.c \
  --enable-foo=shared
$ make install
```

3. Configure Apache for *later installation* of shared modules:

```
$ ./configure --enable-so
$ make install
```

4. Build and install a *third-party* Apache module, say `mod_foo.c`, into its own DSO `mod_foo.so` *outside of* the Apache source tree using [apxs](#):

```
$ cd /path/to/3rdparty
$ apxs -c mod_foo.c
$ apxs -i -a -n foo mod_foo.la
```

In all cases, once the shared module is compiled, you must use a [LoadModule](#) directive in `httpd.conf` to tell Apache to activate the module.



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

This loading can usually be done in two ways: Automatically by a system program called `ld.so` when an executable program is started or manually from within the executing program via a programmatic system interface to the Unix loader through the system calls `dlopen()`/`dlsym()`.

In the first way the DSO's are usually called *shared libraries* or *DSO libraries* and named `libfoo.so` or `libfoo.so.1.2`. They reside in a system directory (usually `/usr/lib`) and the link to the executable program is established at build-time by specifying `-lfoo` to the linker command. This hard-codes library references into the executable program file so that at start-time the Unix loader is able to locate `libfoo.so` in `/usr/lib`, in paths hard-coded via linker-options like `-R` or in paths configured via the environment variable `LD_LIBRARY_PATH`. It then resolves any (yet unresolved) symbols in the executable program which are available in the DSO.

Symbols in the executable program are usually not referenced by the DSO (because it's a reusable library of general code) and hence no further resolving has to be done. The executable program has no need to do anything on its own to use the symbols from the DSO because the complete resolving is done by the Unix loader. (In fact, the code to invoke `ld.so` is part of the run-time startup code which is linked into every executable program which has been bound non-static). The advantage of dynamic loading of common library code is obvious: the library code needs to be stored only once, in a system library like `libc.so`, saving disk space for every program.

In the second way the DSO's are usually called *shared objects* or *DSO files* and can be named with an arbitrary extension (although the canonical name is `foo.so`). These files usually stay inside a program-specific directory and there is no automatically established link to the executable program where they are used. Instead the executable program manually loads the DSO at run-time into its address space via `dlopen()`. At this time no resolving of symbols from the DSO for the executable program is done. But instead the Unix loader automatically resolves any (yet unresolved) symbols in the DSO from the set of symbols exported by the executable program and its already loaded DSO libraries (especially all symbols from the ubiquitous `libc.so`). This way the DSO gets knowledge of the executable program's symbol set as if it had been statically linked with it in the first place.

Finally, to take advantage of the DSO's API the executable program has to resolve particular symbols from the DSO via `dlsym()` for later use inside dispatch tables *etc.* In other words: The executable program has to manually resolve every symbol it needs to be able to use it. The advantage of such a mechanism is that optional program parts need not be loaded (and thus do not spend memory) until they are needed by the program in question. When required, these program parts can be loaded dynamically to extend the base program's functionality.

Although this DSO mechanism sounds straightforward there is at least one difficult step here: The resolving of symbols from the executable program for the DSO when using a DSO to extend a program (the second way). Why? Because "reverse resolving" DSO symbols from the executable program's symbol set is against the library design (where the library has no knowledge about the programs it is used by) and is neither available under all platforms nor standardized. In practice the executable program's global symbols are often not re-exported and thus not available for use in a DSO. Finding a way to force the linker to export all global symbols is the main problem one has to solve when using DSO for extending a program at run-time.

The shared library approach is the typical one, because it is what the DSO mechanism was designed for, hence it is used for nearly all types of libraries the operating system provides. On the other hand using shared objects for extending a program is not used by a lot of programs.

As of 1998 there are only a few software packages available which use the DSO mechanism to actually extend their functionality at run-time: Perl 5 (via its XS mechanism and the DynaLoader module), Netscape Server, *etc.* Starting with version 1.3, Apache joined the crew, because Apache already uses a module concept to extend its functionality and internally uses a dispatch-list-based approach to link external modules into the Apache core functionality. So, Apache is really predestined for using DSO to load its modules at run-time.



The above DSO based features have the following advantages:

- The server package is more flexible at run-time because the actual server process can be assembled at run-time via `LoadModule` `httpd.conf` configuration commands instead of `configure` options at build-time. For instance this way one is able to run different server instances (standard & SSL version, minimalistic & powered up version [`mod_perl`, `PHP3`], *etc.*) with only one Apache installation.
- The server package can be easily extended with third-party modules even after installation. This is at least a great benefit for vendor package maintainers who can create a Apache core package and additional packages containing extensions like `PHP3`, `mod_perl`, `mod_fastcgi`, *etc.*
- Easier Apache module prototyping because with the `DSO/apxs` pair you can both work outside the Apache source tree and only need an `apxs -i` command followed by an `apachectl restart` to bring a new version of your currently developed module into the running Apache server.

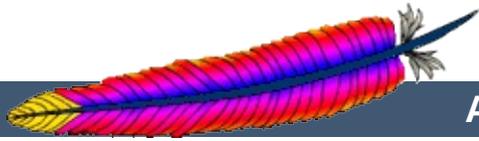
DSO has the following disadvantages:

- The DSO mechanism cannot be used on every platform because not all operating systems support dynamic loading of code into the address space of a program.
- The server is approximately 20% slower at startup time because of the symbol resolving overhead the Unix loader now has to do.
- The server is approximately 5% slower at execution time under some platforms because position independent code (PIC) sometimes needs complicated assembler tricks for relative addressing which are not necessarily as fast as absolute addressing.
- Because DSO modules cannot be linked against other DSO-based libraries (`ld -lfoo`) on all platforms (for instance a.out-based platforms usually don't provide this functionality while ELF-based

platforms do) you cannot use the DSO mechanism for all types of modules. Or in other words, modules compiled as DSO files are restricted to only use symbols from the Apache core, from the C library (`libc`) and all other dynamic or static libraries used by the Apache core, or from static library archives (`libfoo.a`) containing position independent code. The only chances to use other code is to either make sure the Apache core itself already contains a reference to it or loading the code yourself via `dlopen()`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Content Negotiation

Apache supports content negotiation as described in the HTTP/1.1 specification. It can choose the best representation of a resource based on the browser-supplied preferences for media type, languages, character set and encoding. It also implements a couple of features to give more intelligent handling of requests from browsers that send incomplete negotiation information.

Content negotiation is provided by the [mod_negotiation](#) module, which is compiled in by default.



Accept-Content-Negotiation

A resource may be available in several different representations. For example, it might be available in different languages or different media types, or a combination. One way of selecting the most appropriate choice is to give the user an index page, and let them select. However it is often possible for the server to choose automatically. This works because browsers can send, as part of each request, information about what representations they prefer. For example, a browser could indicate that it would like to see information in French, if possible, else English will do. Browsers indicate their preferences by headers in the request. To request only French representations, the browser would send

```
Accept-Language: fr
```

Note that this preference will only be applied when there is a choice of representations and they vary by language.

As an example of a more complex request, this browser has been configured to accept French and English, but prefer French, and to accept various media types, preferring HTML over plain text or other text types, and preferring GIF or JPEG over other media types, but also allowing any other media type as a last resort:

```
Accept-Language: fr; q=1.0, en; q=0.5  
Accept: text/html; q=1.0, text/*; q=0.8, image/gif; q=0.6, image/jpeg;  
q=0.6, image/*; q=0.5, */*; q=0.1
```

Apache supports 'server driven' content negotiation, as defined in the HTTP/1.1 specification. It fully supports the Accept, Accept - Language, Accept - Charset and Accept - Encoding request headers. Apache also supports 'transparent' content negotiation, which is an experimental negotiation protocol defined in RFC 2295 and RFC 2296. It does not offer support for 'feature negotiation' as defined in these RFCs.

A **resource** is a conceptual entity identified by a URI (RFC 2396). An

HTTP server like Apache provides access to **representations** of the resource(s) within its namespace, with each representation in the form of a sequence of bytes with a defined media type, character set, encoding, etc. Each resource may be associated with zero, one, or more than one representation at any given time. If multiple representations are available, the resource is referred to as **negotiable** and each of its representations is termed a **variant**. The ways in which the variants for a negotiable resource vary are called the **dimensions** of negotiation.



Negotiation in Apache

In order to negotiate a resource, the server needs to be given information about each of the variants. This is done in one of two ways:

- Using a type map (*i.e.*, a *.var file) which names the files containing the variants explicitly, or
- Using a 'MultiViews' search, where the server does an implicit filename pattern match and chooses from among the results.

Using a type-map file

A type map is a document which is associated with the handler named type-map (or, for backwards-compatibility with older Apache configurations, the [MIME-type](#) application/x-type-map). Note that to use this feature, you must have a handler set in the configuration that defines a file suffix as type-map; this is best done with

```
AddHandler type-map .var
```

in the server configuration file.

Type map files should have the same name as the resource which they are describing, and have an entry for each available variant; these entries consist of contiguous HTTP-format header lines. Entries for different variants are separated by blank lines. Blank lines are illegal within an entry. It is conventional to begin a map file with an entry for the combined entity as a whole (although this is not required, and if present will be ignored). An example map file is shown below. This file would be named foo.var, as it describes a resource named foo.

```
URI: foo

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

URI: foo.fr.de.html
```

```
Content-type: text/html;charset=iso-8859-2
Content-language: fr, de
```

Note also that a typemap file will take precedence over the filename's extension, even when Multiviews is on. If the variants have different source qualities, that may be indicated by the "qs" parameter to the media type, as in this picture (available as JPEG, GIF, or ASCII-art):

```
URI: foo

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

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

URI: foo.txt
Content-type: text/plain; qs=0.01
```

qs values can vary in the range 0.000 to 1.000. Note that any variant with a qs value of 0.000 will never be chosen. Variants with no 'qs' parameter value are given a qs factor of 1.0. The qs parameter indicates the relative 'quality' of this variant compared to the other available variants, independent of the client's capabilities. For example, a JPEG file is usually of higher source quality than an ASCII file if it is attempting to represent a photograph. However, if the resource being represented is an original ASCII art, then an ASCII representation would have a higher source quality than a JPEG representation. A qs value is therefore specific to a given variant depending on the nature of the resource it represents.

The full list of headers recognized is available in the [mod_negotiation typemap](#) documentation.

Multiviews

MultiViews is a per-directory option, meaning it can be set with an [Options](#) directive within a [<Directory>](#), [<Location>](#) or [<Files>](#)

section in `httpd.conf`, or (if [AllowOverride](#) is properly set) in `.htaccess` files. Note that `Options All` does not set `MultiViews`; you have to ask for it by name.

The effect of `MultiViews` is as follows: if the server receives a request for `/some/dir/foo`, if `/some/dir` has `MultiViews` enabled, and `/some/dir/foo` does *not* exist, then the server reads the directory looking for files named `foo.*`, and effectively fakes up a type map which names all those files, assigning them the same media types and content-encodings it would have if the client had asked for one of them by name. It then chooses the best match to the client's requirements.

`MultiViews` may also apply to searches for the file named by the [DirectoryIndex](#) directive, if the server is trying to index a directory. If the configuration files specify

```
DirectoryIndex index
```

then the server will arbitrate between `index.html` and `index.html3` if both are present. If neither are present, and `index.cgi` is there, the server will run it.

If one of the files found when reading the directory does not have an extension recognized by `mod_mime` to designate its Charset, Content-Type, Language, or Encoding, then the result depends on the setting of the [MultiViewsMatch](#) directive. This directive determines whether handlers, filters, and other extension types can participate in `MultiViews` negotiation.



After Apache has obtained a list of the variants for a given resource, either from a type-map file or from the filenames in the directory, it invokes one of two methods to decide on the 'best' variant to return, if any. It is not necessary to know any of the details of how negotiation actually takes place in order to use Apache's content negotiation features. However the rest of this document explains the methods used for those interested.

There are two negotiation methods:

1. **Server driven negotiation with the Apache algorithm** is used in the normal case. The Apache algorithm is explained in more detail below. When this algorithm is used, Apache can sometimes 'fiddle' the quality factor of a particular dimension to achieve a better result. The ways Apache can fiddle quality factors is explained in more detail below.
2. **Transparent content negotiation** is used when the browser specifically requests this through the mechanism defined in RFC 2295. This negotiation method gives the browser full control over deciding on the 'best' variant, the result is therefore dependent on the specific algorithms used by the browser. As part of the transparent negotiation process, the browser can ask Apache to run the 'remote variant selection algorithm' defined in RFC 2296.

Dimensions of Negotiation

Dimension Notes

Media Type	Browser indicates preferences with the Accept header field. Each item can have an associated quality factor. Variant description can also have a quality factor (the "qs" parameter).
Language	Browser indicates preferences with the Accept - Language header field. Each item can have a quality factor. Variants

	can be associated with none, one or more than one language.
Encoding	Browser indicates preference with the Accept - Encoding header field. Each item can have a quality factor.
Charset	Browser indicates preference with the Accept - Charset header field. Each item can have a quality factor. Variants can indicate a charset as a parameter of the media type.

Apache Negotiation Algorithm

Apache can use the following algorithm to select the 'best' variant (if any) to return to the browser. This algorithm is not further configurable. It operates as follows:

1. First, for each dimension of the negotiation, check the appropriate *Accept** header field and assign a quality to each variant. If the *Accept** header for any dimension implies that this variant is not acceptable, eliminate it. If no variants remain, go to step 4.
2. Select the 'best' variant by a process of elimination. Each of the following tests is applied in order. Any variants not selected at each test are eliminated. After each test, if only one variant remains, select it as the best match and proceed to step 3. If more than one variant remains, move on to the next test.
 1. Multiply the quality factor from the Accept header with the quality-of-source factor for this variants media type, and select the variants with the highest value.
 2. Select the variants with the highest language quality factor.
 3. Select the variants with the best language match, using either the order of languages in the Accept - Language header (if present), or else the order of languages in the LanguagePriority directive (if present).
 4. Select the variants with the highest 'level' media parameter

(used to give the version of text/html media types).

5. Select variants with the best charset media parameters, as given on the Accept - Charset header line. Charset ISO-8859-1 is acceptable unless explicitly excluded. Variants with a text/* media type but not explicitly associated with a particular charset are assumed to be in ISO-8859-1.
 6. Select those variants which have associated charset media parameters that are *not* ISO-8859-1. If there are no such variants, select all variants instead.
 7. Select the variants with the best encoding. If there are variants with an encoding that is acceptable to the user-agent, select only these variants. Otherwise if there is a mix of encoded and non-encoded variants, select only the unencoded variants. If either all variants are encoded or all variants are not encoded, select all variants.
 8. Select the variants with the smallest content length.
 9. Select the first variant of those remaining. This will be either the first listed in the type-map file, or when variants are read from the directory, the one whose file name comes first when sorted using ASCII code order.
3. The algorithm has now selected one 'best' variant, so return it as the response. The HTTP response header Vary is set to indicate the dimensions of negotiation (browsers and caches can use this information when caching the resource). End.
 4. To get here means no variant was selected (because none are acceptable to the browser). Return a 406 status (meaning "No acceptable representation") with a response body consisting of an HTML document listing the available variants. Also set the HTTP Vary header to indicate the dimensions of variance.



Handling With Quality Values

Apache sometimes changes the quality values from what would be expected by a strict interpretation of the Apache negotiation algorithm above. This is to get a better result from the algorithm for browsers which do not send full or accurate information. Some of the most popular browsers send Accept header information which would otherwise result in the selection of the wrong variant in many cases. If a browser sends full and correct information these fiddles will not be applied.

Media Types and Wildcards

The Accept : request header indicates preferences for media types. It can also include 'wildcard' media types, such as "image/*" or "*/*" where the * matches any string. So a request including:

```
Accept: image/*, */*
```

would indicate that any type starting "image/" is acceptable, as is any other type. Some browsers routinely send wildcards in addition to explicit types they can handle. For example:

```
Accept: text/html, text/plain, image/gif, image/jpeg, */*
```

The intention of this is to indicate that the explicitly listed types are preferred, but if a different representation is available, that is ok too. Using explicit quality values, what the browser really wants is something like:

```
Accept: text/html, text/plain, image/gif, image/jpeg, */*; q=0.01
```

The explicit types have no quality factor, so they default to a preference of 1.0 (the highest). The wildcard */* is given a low preference of 0.01, so other types will only be returned if no variant matches an explicitly listed type.

If the `Accept :` header contains *no* q factors at all, Apache sets the q value of `*/*`, if present, to 0.01 to emulate the desired behavior. It also sets the q value of wildcards of the format `type/*` to 0.02 (so these are preferred over matches against `*/*`). If any media type on the `Accept :` header contains a q factor, these special values are *not* applied, so requests from browsers which send the explicit information to start with work as expected.

Language Negotiation Exceptions

New in Apache 2.0, some exceptions have been added to the negotiation algorithm to allow graceful fallback when language negotiation fails to find a match.

When a client requests a page on your server, but the server cannot find a single page that matches the `Accept - language` sent by the browser, the server will return either a "No Acceptable Variant" or "Multiple Choices" response to the client. To avoid these error messages, it is possible to configure Apache to ignore the `Accept - language` in these cases and provide a document that does not explicitly match the client's request. The [ForceLanguagePriority](#) directive can be used to override one or both of these error messages and substitute the server's judgement in the form of the [LanguagePriority](#) directive.

The server will also attempt to match language-subsets when no other match can be found. For example, if a client requests documents with the language `en-GB` for British English, the server is not normally allowed by the HTTP/1.1 standard to match that against a document that is marked as simply `en`. (Note that it is almost surely a configuration error to include `en-GB` and not `en` in the `Accept - Language` header, since it is very unlikely that a reader understands British English, but doesn't understand English in general. Unfortunately, many current clients have default configurations that resemble this.) However, if no other language match is possible and the server is about to return a "No Acceptable Variants"

error or fallback to the [LanguagePriority](#), the server will ignore the subset specification and match en-GB against en documents. Implicitly, Apache will add the parent language to the client's acceptable language list with a very low quality value. But note that if the client requests "en-GB; q=0.9, fr; q=0.8", and the server has documents designated "en" and "fr", then the "fr" document will be returned. This is necessary to maintain compliance with the HTTP/1.1 specification and to work effectively with properly configured clients.

In order to support advanced techniques (such as cookies or special URL-paths) to determine the user's preferred language, since Apache 2.0.47 [mod_negotiation](#) recognizes the [environment variable](#) `prefer-language`. If it exists and contains an appropriate language tag, [mod_negotiation](#) will try to select a matching variant. If there's no such variant, the normal negotiation process applies.

Example

```
SetEnvIf Cookie "language=(.+)" prefer-language=$1
Header append Vary cookie
```



Extensions to Transparent Content Negotiation

Apache extends the transparent content negotiation protocol (RFC 2295) as follows. A new `{encoding . . .}` element is used in variant lists to label variants which are available with a specific content-encoding only. The implementation of the RVSA/1.0 algorithm (RFC 2296) is extended to recognize encoded variants in the list, and to use them as candidate variants whenever their encodings are acceptable according to the `Accept-Encoding` request header. The RVSA/1.0 implementation does not round computed quality factors to 5 decimal places before choosing the best variant.



File naming, mime and naming conventions

If you are using language negotiation you can choose between different naming conventions, because files can have more than one extension, and the order of the extensions is normally irrelevant (see the [mod_mime](#) documentation for details).

A typical file has a MIME-type extension (e.g., `html`), maybe an encoding extension (e.g., `gz`), and of course a language extension (e.g., `en`) when we have different language variants of this file.

Examples:

- `foo.en.html`
- `foo.html.en`
- `foo.en.html.gz`

Here some more examples of filenames together with valid and invalid hyperlinks:

Filename	Valid hyperlink	Invalid hyperlink
<i>foo.html.en</i>	foo foo.html	-
<i>foo.en.html</i>	foo	foo.html
<i>foo.html.en.gz</i>	foo foo.html	foo.gz foo.html.gz
<i>foo.en.html.gz</i>	foo	foo.html foo.html.gz foo.gz
<i>foo.gz.html.en</i>	foo foo.gz foo.gz.html	foo.html
<i>foo.html.gz.en</i>	foo foo.html foo.html.gz	foo.gz

Looking at the table above, you will notice that it is always possible to use the name without any extensions in a hyperlink (*e.g.*, `foo`). The advantage is that you can hide the actual type of a document resp. file and can change it later, *e.g.*, from `html` to `shtml` or `cgi` without changing any hyperlink references.

If you want to continue to use a MIME-type in your hyperlinks (*e.g.* `foo.html`) the language extension (including an encoding extension if there is one) must be on the right hand side of the MIME-type extension (*e.g.*, `foo.html.en`).

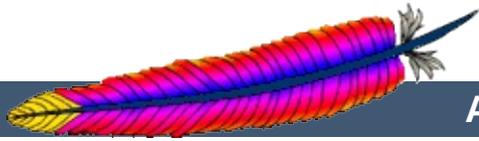


Non-Caching

When a cache stores a representation, it associates it with the request URL. The next time that URL is requested, the cache can use the stored representation. But, if the resource is negotiable at the server, this might result in only the first requested variant being cached and subsequent cache hits might return the wrong response. To prevent this, Apache normally marks all responses that are returned after content negotiation as non-cacheable by HTTP/1.0 clients. Apache also supports the HTTP/1.1 protocol features to allow caching of negotiated responses.

For requests which come from a HTTP/1.0 compliant client (either a browser or a cache), the directive [CacheNegotiatedDocs](#) can be used to allow caching of responses which were subject to negotiation. This directive can be given in the server config or virtual host, and takes no arguments. It has no effect on requests from HTTP/1.1 clients.

For HTTP/1.1 clients, Apache sends a Vary HTTP response header to indicate the negotiation dimensions for the response. Caches can use this information to determine whether a subsequent request can be served from the local copy. To encourage a cache to use the local copy regardless of the negotiation dimensions, set the [force-no-vary environment variable](#).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Custom Error Responses

Additional functionality allows webmasters to configure the response of Apache to some error or problem.

Customizable responses can be defined to be activated in the event of a server detected error or problem.

If a script crashes and produces a "500 Server Error" response, then this response can be replaced with either some friendlier text or by a redirection to another URL (local or external).



Old Behavior

NCSA httpd 1.3 would return some boring old error/problem message which would often be meaningless to the user, and would provide no means of logging the symptoms which caused it.

New Behavior

The server can be asked to:

1. Display some other text, instead of the NCSA hard coded messages, or
2. redirect to a local URL, or
3. redirect to an external URL.

Redirecting to another URL can be useful, but only if some information can be passed which can then be used to explain and/or log the error/problem more clearly.

To achieve this, Apache will define new CGI-like environment variables:

```
REDIRECT_HTTP_ACCEPT=*/, image/gif, image/x-xbitmap, image/jpeg
REDIRECT_HTTP_USER_AGENT=Mozilla/1.1b2 (X11; I; HP-UX A.09.05
9000/712)
REDIRECT_PATH=./bin:/usr/local/bin:/etc
REDIRECT_QUERY_STRING=
REDIRECT_REMOTE_ADDR=121.345.78.123
REDIRECT_REMOTE_HOST=ooh.ahhh.com
REDIRECT_SERVER_NAME=crash.bang.edu
REDIRECT_SERVER_PORT=80
REDIRECT_SERVER_SOFTWARE=Apache/0.8.15
REDIRECT_URL=/cgi-bin/buggy.pl
```

Note the REDIRECT_ prefix.

At least REDIRECT_URL and REDIRECT_QUERY_STRING will be passed to the new URL (assuming it's a cgi-script or a cgi-include). The other

variables will exist only if they existed prior to the error/problem. **None** of these will be set if your [ErrorDocument](#) is an *external* redirect (anything starting with a scheme name like `http:`, even if it refers to the same host as the server).



Use of [ErrorDocument](#) is enabled for .htaccess files when the [AllowOverride](#) is set accordingly.

Here are some examples...

```
ErrorDocument 500 /cgi-bin/crash-recover
ErrorDocument 500 "Sorry, our script crashed. Oh dear"
ErrorDocument 500 http://xxx/
ErrorDocument 404 /Lame_excuses/not_found.html
ErrorDocument 401 /Subscription/how_to_subscribe.html
```

The syntax is,

```
ErrorDocument <3-digit-code> <action>
```

where the action can be,

1. Text to be displayed. Wrap the text with quotes (").
2. An external URL to redirect to.
3. A local URL to redirect to.



Apache's behavior to redirected URLs has been modified so that additional environment variables are available to a script/server-include.

Old behavior

Standard CGI vars were made available to a script which has been redirected to. No indication of where the redirection came from was provided.

New behavior

A new batch of environment variables will be initialized for use by a script which has been redirected to. Each new variable will have the prefix `REDIRECT_`. `REDIRECT_` environment variables are created from the CGI environment variables which existed prior to the redirect, they are renamed with a `REDIRECT_` prefix, *i.e.*, `HTTP_USER_AGENT` becomes `REDIRECT_HTTP_USER_AGENT`. In addition to these new variables, Apache will define `REDIRECT_URL` and `REDIRECT_STATUS` to help the script trace its origin. Both the original URL and the URL being redirected to can be logged in the access log.

If the `ErrorDocument` specifies a local redirect to a CGI script, the script should include a `"Status:"` header field in its output in order to ensure the propagation all the way back to the client of the error condition that caused it to be invoked. For instance, a Perl `ErrorDocument` script might include the following:

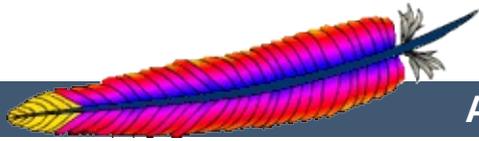
```
...
print "Content-type: text/html\n";
printf "Status: %s Condition Intercepted\n", $ENV{"REDIRECT_STATUS"};
...
```

If the script is dedicated to handling a particular error condition, such as `404 Not Found`, it can use the specific code and error text instead.

Note that the script *must* emit an appropriate `Status:` header (such as `302 Found`), if the response contains a `Location:` header (in order to issue a client side redirect). Otherwise the `Location:` header may have no effect.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Anbindung

Konfiguration der vom Apache verwendeten Adressen und Ports.

Siehe auch

[Virtuelle Hosts](#)

[Probleme bezüglich DNS und Apache](#)



Referenzierte Module	Referenzierte Direktiven
<u>core</u>	<u><VirtualHost></u>
<u>mpm_common</u>	<u>Listen</u>

Beim Start bindet sich der Apache an bestimmte Adressen und Ports der lokalen Maschine und wartet auf eingehende Anfragen. Standardmäßig lauscht er an allen Adressen des Systems. Es kann ihm jedoch mitgeteilt werden, an bestimmten Ports zu lauschen oder nur an ausgewählten Adressen, bzw. einer Kombination aus beidem. Dies wird oft mit der Funktionalität virtueller Hosts kombiniert, die bestimmt, wie der Apache auf verschiedene IP-Adressen, Hostnamen und Ports reagiert.

Die Direktive Listen weist den Server an, eingehende Anfragen nur an bestimmten Ports oder Adress/Port-Kombinationen zu akzeptieren. Wenn bei der Listen-Direktive nur eine Portnummer angegeben wird, dann lauscht der Server auf allen Netzwerkinterfaces an dem angegebenen Port. Ist auch eine IP-Adresse angegeben, dann lauscht der Server an der angegebenen Schnittstelle auf dem angegebenen Port. Es können mehrere Listen-Anweisungen verwendet werden, um eine Reihe von Adressen und Ports anzugeben, an denen gelauscht werden soll. Der Server wird dann auf Anfragen an jeder der abgehörten Adressen und Ports antworten.

Um beispielsweise den Server zu veranlassen, auf allen Netzwerkinterfaces sowohl an Port 80, als auch an Port 8000 Verbindungen zu akzeptieren, geben Sie an:

```
Listen 80
Listen 8000
```

Um den Server Verbindungen auf Port 80 eines Interfaces und auf Port 8000 eines anderen akzeptieren zu lassen, geben Sie an:

```
Listen 192.0.2.1:80  
Listen 192.0.2.5:8000
```

IPv6-Adressen müssen wie im folgenden Beispiel in eckigen Klammern angegeben werden:

```
Listen [2001:db8::a00:20ff:fea7:ccea]:80
```



Eine wachsende Anzahl von Plattformen implementiert IPv6. Die [APR](#) unterstützt IPv6 auf den meisten dieser Plattformen und ermöglicht dem Apache, IPv6-Sockets zu verwenden und Anfragen zu behandeln, die über IPv6 gesendet wurden.

Für Apache-Administratoren kommt erschwerend die Frage hinzu, ob IPv6-Sockets sowohl IPv4- als auch IPv6-Verbindungen handhaben können. Zum Betrieb von IPv4-Verbindungen an IPv6-Sockets werden auf IPv6 abgebildete IPv4-Adressen (*Anm.d.Ü.: so genannete IPv4-gemappede IPv6-Adressen*) verwendet, welche standardmäßig auf den meisten Plattformen erlaubt sind. Unter FreeBSD, NetBSD und OpenBSD jedoch sind sie standardmäßig deaktiviert, um den Systemgrundsätzen dieser Plattformen zu entsprechen. Doch selbst auf Systemen, wo dies standardmäßig deaktiviert ist, kann dieses Verhalten mit einem speziellen [configure](#)-Parameter für den Apache geändert werden.

Auf der anderen Seite ist die Verwendung von gemappten Adressen bei einigen Plattformen wie Linux und True64 der **einzige** Weg, sowohl IPv4 wie auch IPv6 zu verwenden. Wenn Sie möchten, dass der Apache IPv4- und IPv6-Verbindungen mit einem Minimum an Sockets behandelt, was die Verwendung von IPv4-gemappten IPv6-Adressen erfordert, dann müssen Sie die [configure](#)-Option `--enable-v4-mapped` angeben.

`--enable-v4-mapped` ist die Voreinstellung auf allen Plattformen außer FreeBSD, NetBSD und OpenBSD, so dass Ihr Apache wahrscheinlich so übersetzt wurde.

Geben Sie wie in dem folgenden Beispiel bei allen [Listen](#)-Anweisungen eine IPv4-Adresse an, wenn Sie möchten, dass Ihr Apache lediglich IPv4-Adressen behandelt, unabhängig davon, was Ihre Plattform und die APR unterstützen:

```
Listen 0.0.0.0:80
```

Listen 192.0.2.1:80

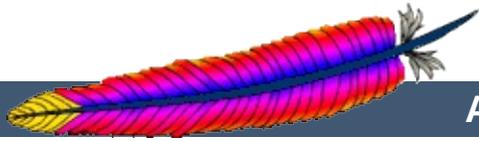
Wenn Sie möchten, dass der Apache IPv4- und IPv6-Verbindungen an separaten Sockets behandelt (d.h. IPv4-gemapped Adressen deaktiviert werden sollen) und Ihre Plattform es unterstützt, dann müssen Sie die [configure](#)-Option `--disable-v4-mapped` angeben. Unter FreeBSD, NetBSD und OpenBSD ist `--disable-v4-mapped` voreingestellt.



Listen implementiert keine virtuellen Hosts. Es teilt dem Hauptserver lediglich mit, an welchen Adressen und Ports er zu lauschen hat. Werden keine `<VirtualHost>`-Container verwendet, verhält sich der Server bei allen angenommenen Anfragen gleich. `<VirtualHost>`-Abschnitte können jedoch dazu verwendet werden, ein unterschiedliches Verhalten für eine oder mehrere Adressen und Ports festzulegen. Um einen virtuellen Host einzurichten, muss dem Server zunächst mitgeteilt werden, an den betreffenden Adressen und Ports zu lauschen. Dann sollte ein `<VirtualHost>`-Abschnitt für eine bestimmte Adresse und einen Port erstellt werden, um das Verhalten dieses virtuellen Hosts festzulegen. Beachten Sie bitte, dass auf einen `<VirtualHost>` nicht zugegriffen werden kann, wenn er für eine Adresse und einen Port eingerichtet wurde, an dem der Server nicht lauscht.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Multi-Processing-Module (MPMs)

Das Dokument beschreibt, was ein Multi-Processing-Modul ist und wie solche Module beim Apache HTTP Server verwendet werden.



Der Apache HTTP Server wurde als leistungsfähiger und flexibler Webserver konzipiert, der auf einer Vielzahl von Plattformen in einer Reihe unterschiedlicher Umgebungen arbeiten kann. Unterschiedliche Plattformen und unterschiedliche Umgebungen verlangen oftmals verschiedene Fähigkeiten oder kennen verschiedene Wege, die gleiche Funktionalität sehr effizient zu implementieren. Der Apache hat durch seinen modularen Aufbau schon immer eine breite Auswahl von Umgebungen unterstützt. Dieses Design erlaubt es dem Webmaster, durch Auswahl der Module, die zur Kompilierungszeit oder zur Laufzeit geladen werden, die Features auszuwählen, die in den Server integriert werden.

Der Apache 2.0 erweitert dieses modulare Design auf die grundlegenden Funktionen eines Webserver. Der Server wird mit einer Auswahl von Multi-Processing-Modulen (MPMs) ausgeliefert, die für die Bindung an Netzwerkports der Maschine, die Annahme von Anfragen und die Abfertigung von Kindprozessen zur Behandlung der Anfragen zuständig sind.

Die Erweiterung des modularen Aufbaus auf diese Ebene des Servers bringt zwei wesentliche Vorteile:

- Der Apache kann nun eine Vielfalt von Betriebssystemen sauberer und effizienter unterstützen. Insbesondere die Windows-Version des Apache ist jetzt deutlich effizienter, da mpm_winnt native Netzwerkfähigkeiten anstelle der im Apache 1.3 verwendeten POSIX-Schicht benutzen kann. Dieser Vorteil gilt auch für andere Betriebssysteme, für die spezielle MPMs implementiert sind.
- Der Server läßt sich besser auf die Bedürfnisse der jeweiligen Website anpassen. Sites beispielsweise, die eine hohe Skalierbarkeit benötigen, können ein Threaded-MPM wie worker oder event wählen, während Sites, die Stabilität oder Kompatibilität mit älterer Software erfordern, prefork wählen können.

Auf Anwendererebene erscheinen MPMs fast wie andere Apache-Module. Der Hauptunterschied ist, dass jeweils nur ein einziges MPM in den Server geladen werden kann. Die Liste der verfügbaren MPMs finden Sie im [Modul-Index](#).



MPMs müssen während der (*Anm.d.Ü.: Quelltext-*)Konfiguration ausgewählt und in den Server einkompiliert werden. Compiler sind in der Lage eine Reihe von Funktionen zu optimieren, wenn Threads verwendet werden. Sie können dies allerdings nur, wenn sie wissen, dass Threads benutzt werden.

Um das gewünschte MPM tatsächlich auszuwählen, verwenden Sie beim [configure](#)-Skript das Argument `--with-mpm=NAME`. *NAME* ist der Name des gewünschten MPMs.

Ist der Server kompiliert, so ist es mittels `./httpd -l` möglich, das ausgewählte MPM zu ermitteln. Dieser Befehl listet alle in den Server einkompilierten Module auf, einschließlich des MPM.



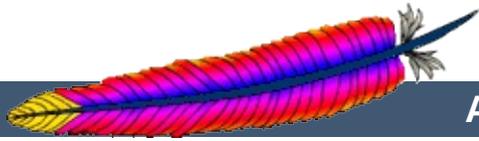
MPM Voreinstellungen

Die folgende Tabelle gibt die voreingestellten MPMs für verschiedene Betriebssysteme an. Wenn Sie während der Kompilierung keine andere Auswahl treffen, wird dieses MPM gewählt.

BeOS	<code>beos</code>
Netware	<code>mpm_netware</code>
OS/2	<code>mpmt_os2</code>
Unix	<code>prefork</code>
Windows	<code>mpm_winnt</code>

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Environment Variables in Apache

The Apache HTTP Server provides a mechanism for storing information in named variables that are called *environment variables*. This information can be used to control various operations such as logging or access control. The variables are also used as a mechanism to communicate with external programs such as CGI scripts. This document discusses different ways to manipulate and use these variables.

Although these variables are referred to as *environment variables*, they are not the same as the environment variables controlled by the underlying operating system. Instead, these variables are stored and manipulated in an internal Apache structure. They only become actual operating system environment variables when they are provided to CGI scripts and Server Side Include scripts. If you wish to manipulate the operating system environment under which the server itself runs, you must use the standard environment manipulation mechanisms provided by your operating system shell.



Related Modules	Related Directives
mod_env	BrowserMatch
mod_rewrite	BrowserMatchNoCase
mod_setenvif	PassEnv
mod_unique_id	RewriteRule
	SetEnv
	SetEnvIf
	SetEnvIfNoCase
	UnsetEnv

Basic Environment Manipulation

The most basic way to set an environment variable in Apache is using the unconditional [SetEnv](#) directive. Variables may also be passed from the environment of the shell which started the server using the [PassEnv](#) directive.

Conditional Per-Request Settings

For additional flexibility, the directives provided by [mod_setenvif](#) allow environment variables to be set on a per-request basis, conditional on characteristics of particular requests. For example, a variable could be set only when a specific browser (User-Agent) is making a request, or only when a specific Referer [sic] header is found. Even more flexibility is available through the [mod_rewrite](#)'s [RewriteRule](#) which uses the `[E= . . .]` option to set environment variables.

Unique Identifiers

Finally, [mod_unique_id](#) sets the environment variable `UNIQUE_ID` for each request to a value which is guaranteed to be unique across "all" requests under very specific conditions.

Standard CGI Variables

In addition to all environment variables set within the Apache configuration and passed from the shell, CGI scripts and SSI pages are provided with a set of environment variables containing meta-information about the request as required by the [CGI specification](#).

Some Caveats

- It is not possible to override or change the standard CGI variables using the environment manipulation directives.
- When [suexec](#) is used to launch CGI scripts, the environment will be cleaned down to a set of *safe* variables before CGI scripts are launched. The list of *safe* variables is defined at compile-time in `suexec.c`.
- For portability reasons, the names of environment variables may contain only letters, numbers, and the underscore character. In addition, the first character may not be a number. Characters which do not match this restriction will be replaced by an underscore when passed to CGI scripts and SSI pages.
- The [SetEnv](#) directive runs late during request processing meaning that directives such as [SetEnvIf](#) and [RewriteCond](#) will not see the variables set with it.



Related Modules	Related Directives
mod_authz_host	Allow
mod_cgi	CustomLog
mod_ext_filter	Deny
mod_headers	ExtFilterDefine
mod_include	Header
mod_log_config	LogFormat
mod_rewrite	RewriteCond
	RewriteRule

CGI Scripts

One of the primary uses of environment variables is to communicate information to CGI scripts. As discussed above, the environment passed to CGI scripts includes standard meta-information about the request in addition to any variables set within the Apache configuration. For more details, see the [CGI tutorial](#).

SSI Pages

Server-parsed (SSI) documents processed by [mod_include](#)'s INCLUDES filter can print environment variables using the echo element, and can use environment variables in flow control elements to make parts of a page conditional on characteristics of a request. Apache also provides SSI pages with the standard CGI environment variables as discussed above. For more details, see the [SSI tutorial](#).

Access Control

Access to the server can be controlled based on the value of environment variables using the `allow from env=` and `deny from env=` directives. In combination with [SetEnvIf](#), this allows for flexible control of access to the server based on characteristics of the client. For

example, you can use these directives to deny access to a particular browser (User-Agent).

Conditional Logging

Environment variables can be logged in the access log using the [LogFormat](#) option %e. In addition, the decision on whether or not to log requests can be made based on the status of environment variables using the conditional form of the [CustomLog](#) directive. In combination with [SetEnvIf](#) this allows for flexible control of which requests are logged. For example, you can choose not to log requests for filenames ending in gif, or you can choose to only log requests from clients which are outside your subnet.

Conditional Response Headers

The [Header](#) directive can use the presence or absence of an environment variable to determine whether or not a certain HTTP header will be placed in the response to the client. This allows, for example, a certain response header to be sent only if a corresponding header is received in the request from the client.

External Filter Activation

External filters configured by [mod_ext_filter](#) using the [ExtFilterDefine](#) directive can be activated conditional on an environment variable using the `disableenv=` and `enableenv=` options.

URL Rewriting

The `%{ENV:variable}` form of *TestString* in the [RewriteCond](#) allows [mod_rewrite](#)'s rewrite engine to make decisions conditional on environment variables. Note that the variables accessible in [mod_rewrite](#) without the `ENV:` prefix are not actually environment variables. Rather, they are variables special to [mod_rewrite](#) which

cannot be accessed from other modules.



Interoperability problems have led to the introduction of mechanisms to modify the way Apache behaves when talking to particular clients. To make these mechanisms as flexible as possible, they are invoked by defining environment variables, typically with [BrowserMatch](#), though [SetEnv](#) and [PassEnv](#) could also be used, for example.

downgrade-1.0

This forces the request to be treated as a HTTP/1.0 request even if it was in a later dialect.

force-gzip

If you have the DEFLATE filter activated, this environment variable will ignore the accept-encoding setting of your browser and will send compressed output unconditionally.

force-no-vary

This causes any Vary fields to be removed from the response header before it is sent back to the client. Some clients don't interpret this field correctly; setting this variable can work around this problem. Setting this variable also implies **force-response-1.0**.

force-response-1.0

This forces an HTTP/1.0 response to clients making an HTTP/1.0 request. It was originally implemented as a result of a problem with AOL's proxies. Some HTTP/1.0 clients may not behave correctly when given an HTTP/1.1 response, and this can be used to interoperate with them.

gzip-only-text/html

When set to a value of "1", this variable disables the DEFLATE output filter provided by [mod_deflate](#) for content-types other than

text/html. If you'd rather use statically compressed files, [mod_negotiation](#) evaluates the variable as well (not only for gzip, but for all encodings that differ from "identity").

no-gzip

When set, the DEFLATE filter of [mod_deflate](#) will be turned off and [mod_negotiation](#) will refuse to deliver encoded resources.

no-cache

Available in versions 2.2.12 and later

When set, [mod_cache](#) will not save an otherwise cacheable response. This environment variable does not influence whether a response already in the cache will be served for the current request.

nokeepalive

This disables [KeepAlive](#) when set.

prefer-language

This influences [mod_negotiation](#)'s behaviour. If it contains a language tag (such as en, ja or x-klingon), [mod_negotiation](#) tries to deliver a variant with that language. If there's no such variant, the normal [negotiation](#) process applies.

redirect-carefully

This forces the server to be more careful when sending a redirect to the client. This is typically used when a client has a known problem handling redirects. This was originally implemented as a result of a problem with Microsoft's WebFolders software which has a problem handling redirects on directory resources via DAV methods.

suppress-error-charset

Available in versions after 2.0.54

When Apache issues a redirect in response to a client request, the response includes some actual text to be displayed in case the client can't (or doesn't) automatically follow the redirection. Apache ordinarily labels this text according to the character set which it uses, which is ISO-8859-1.

However, if the redirection is to a page that uses a different character set, some broken browser versions will try to use the character set from the redirection text rather than the actual page. This can result in Greek, for instance, being incorrectly rendered.

Setting this environment variable causes Apache to omit the character set for the redirection text, and these broken browsers will then correctly use that of the destination page.

Security note

Sending error pages without a specified character set may allow a cross-site-scripting attack for existing browsers (MSIE) which do not follow the HTTP/1.1 specification and attempt to "guess" the character set from the content. Such browsers can be easily fooled into using the UTF-7 character set, and UTF-7 content from input data (such as the request-URI) will not be escaped by the usual escaping mechanisms designed to prevent cross-site-scripting attacks.

force-proxy-request-1.0, proxy-nokeepalive, proxy-sendchunked, proxy-sendcl, proxy-chain-auth, proxy-interim-response, proxy-initial-not-pooled

These directives alter the protocol behavior of [mod_proxy](#). See the [mod_proxy](#) and [mod_proxy_http](#) documentation for more details.



Changing protocol behavior with misbehaving clients

Earlier versions recommended that the following lines be included in httpd.conf to deal with known client problems. Since the affected clients are no longer seen in the wild, this configuration is likely no-longer necessary.

```
#
# The following directives modify normal HTTP response behavior.
# The first directive disables keepalive for Netscape 2.x and browsers
# spoof it. There are known problems with these browser implementations
# The second directive is for Microsoft Internet Explorer 4.0b2
# which has a broken HTTP/1.1 implementation and does not properly
# support keepalive when it is used on 301 or 302 (redirect) responses.
#
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0

#
# The following directive disables HTTP/1.1 responses to browsers which
# are in violation of the HTTP/1.0 spec by not being able to grok a
# basic 1.1 response.
#
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

Do not log requests for images in the access log

This example keeps requests for images from appearing in the access log. It can be easily modified to prevent logging of particular directories, or to prevent logging of requests coming from particular hosts.

```
SetEnvIf Request_URI \.gif image-request
SetEnvIf Request_URI \.jpg image-request
SetEnvIf Request_URI \.png image-request
CustomLog logs/access_log common env=!image-request
```

Prevent "Image Theft"

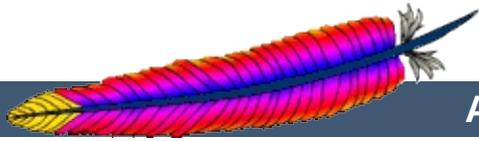
This example shows how to keep people not on your server from using images on your server as inline-images on their pages. This is not a recommended configuration, but it can work in limited circumstances. We assume that all your images are in a directory called /web/images.

```
SetEnvIf Referer "^http://www\.example\.com/" local_referal
# Allow browsers that do not send Referer info
SetEnvIf Referer "^$" local_referal
<Directory /web/images>
    Order Deny,Allow
    Deny from all
    Allow from env=local_referal
</Directory>
```

For more information about this technique, see the "[Keeping Your Images from Adorning Other Sites](#)" tutorial on ServerWatch.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Apache's Handler Use

This document describes the use of Apache's Handlers.



FILES & HANDLERS

Related Modules	Related Directives
mod_actions	Action
mod_asis	AddHandler
mod_cgi	RemoveHandler
mod_imagemap	SetHandler
mod_info	
mod_mime	
mod_negotiation	
mod_status	

A "handler" is an internal Apache representation of the action to be performed when a file is called. Generally, files have implicit handlers, based on the file type. Normally, all files are simply served by the server, but certain file types are "handled" separately.

Handlers may also be configured explicitly, based on either filename extensions or on location, without relation to file type. This is advantageous both because it is a more elegant solution, and because it also allows for both a type **and** a handler to be associated with a file. (See also [Files with Multiple Extensions](#).)

Handlers can either be built into the server or included in a module, or they can be added with the [Action](#) directive. The built-in handlers in the standard distribution are as follows:

- **default-handler**: Send the file using the `default_handler()`, which is the handler used by default to handle static content. (core)
- **send-as-is**: Send file with HTTP headers as is. ([mod_asis](#))
- **cgi-script**: Treat the file as a CGI script. ([mod_cgi](#))
- **imap-file**: Parse as an imagemap rule file. ([mod_imagemap](#))
- **server-info**: Get the server's configuration information. ([mod_info](#))
- **server-status**: Get the server's status report. ([mod_status](#))

- **type-map**: Parse as a type map file for content negotiation.
(mod_negotiation)



Modifying static content using a CGI script

The following directives will cause requests for files with the `html` extension to trigger the launch of the `footer.pl` CGI script.

```
Action add-footer /cgi-bin/footer.pl
AddHandler add-footer .html
```

Then the CGI script is responsible for sending the originally requested document (pointed to by the `PATH_TRANSLATED` environment variable) and making whatever modifications or additions are desired.

Files with HTTP headers

The following directives will enable the `send-as-is` handler, which is used for files which contain their own HTTP headers. All files in the `/web/htdocs/asis/` directory will be processed by the `send-as-is` handler, regardless of their filename extensions.

```
<Directory /web/htdocs/asis>
SetHandler send-as-is
</Directory>
```



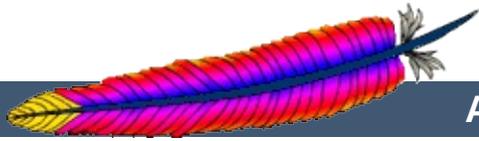
In order to implement the handler features, an addition has been made to the [Apache API](#) that you may wish to make use of. Specifically, a new record has been added to the `request_rec` structure:

```
char *handler
```

If you wish to have your module engage a handler, you need only to set `r->handler` to the name of the handler at any time prior to the `invoke_handler` stage of the request. Handlers are implemented as they were before, albeit using the handler name instead of a content type. While it is not necessary, the naming convention for handlers is to use a dash-separated word, with no slashes, so as to not invade the media type name-space.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

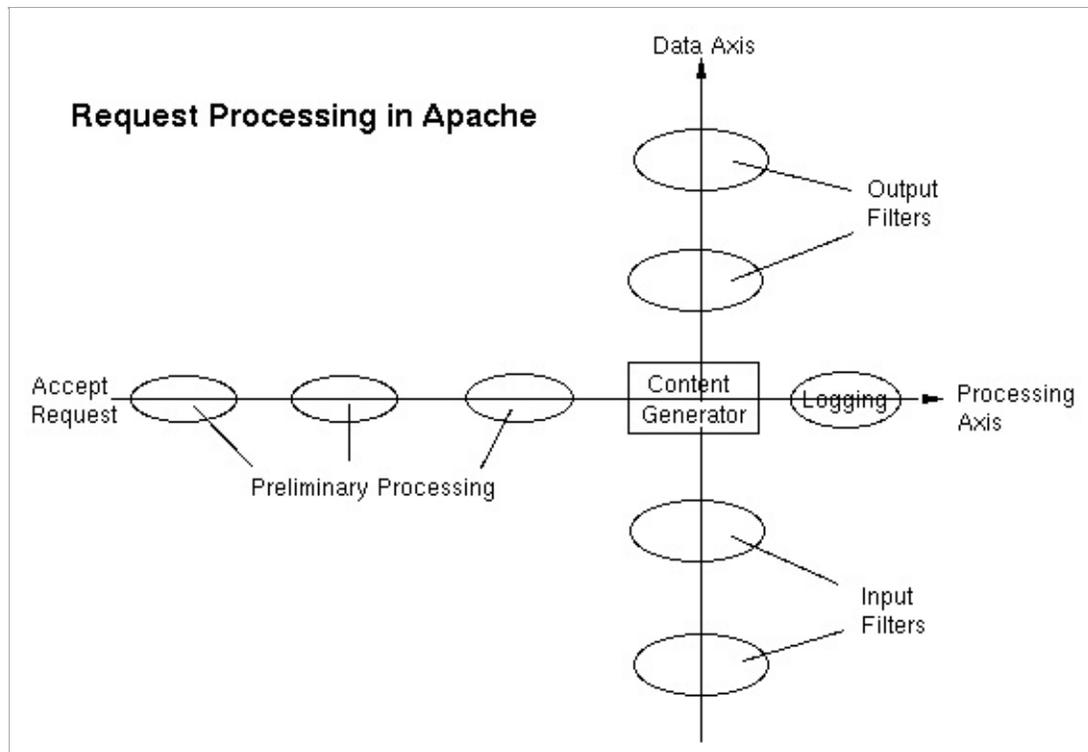
Filters

This document describes the use of filters in Apache.



Related Modules	Related Directives
mod_filter	FilterChain
mod_deflate	FilterDeclare
mod_ext_filter	FilterProtocol
mod_include	FilterProvider
mod_charset_lite	AddInputFilter
	AddOutputFilter
	RemoveInputFilter
	RemoveOutputFilter
	ExtFilterDefine
	ExtFilterOptions
	SetInputFilter
	SetOutputFilter

The Filter Chain is available in Apache 2.0 and higher, and enables applications to process incoming and outgoing data in a highly flexible and configurable manner, regardless of where the data comes from. We can pre-process incoming data, and post-process outgoing data, at will. This is basically independent of the traditional request processing phases.



Some examples of filtering in the standard Apache distribution are:

- [mod_include](#), implements server-side includes.
- [mod_ssl](#), implements SSL encryption (https).
- [mod_deflate](#), implements compression/decompression on the fly.
- [mod_charset_lite](#), transcodes between different character sets.
- [mod_ext_filter](#), runs an external program as a filter.

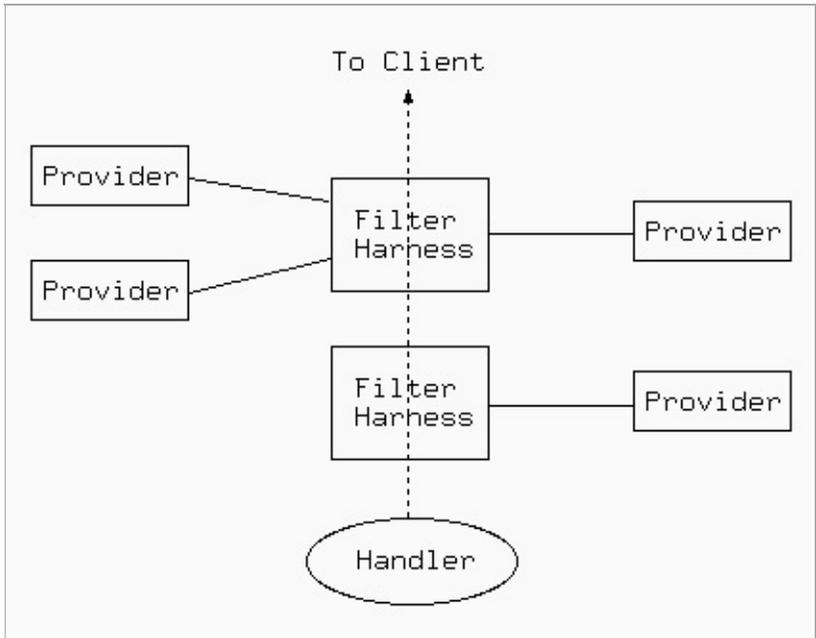
Apache also uses a number of filters internally to perform functions like chunking and byte-range handling.

A wider range of applications are implemented by third-party filter modules available from modules.apache.org and elsewhere. A few of these are:

- HTML and XML processing and rewriting
- XSLT transforms and XIncludes
- XML Namespace support
- File Upload handling and decoding of HTML Forms

- Image processing
- Protection of vulnerable applications such as PHP scripts
- Text search-and-replace editing





mod_filter, included in Apache 2.1 and later, enables the filter chain to be configured dynamically at run time. So for example you can set up a proxy to rewrite HTML with an HTML filter and JPEG images with a completely separate filter, despite the proxy having no prior information about what the origin server will send. This works by using a filter harness, that dispatches to different providers according to the actual contents at runtime. Any filter may be either inserted directly in the chain and run unconditionally, or used as a provider and inserted dynamically. For example,

- an HTML processing filter will only run if the content is text/html or application/xhtml+xml
- A compression filter will only run if the input is a compressible type and not already compressed
- A charset conversion filter will be inserted if a text document is not already in the desired charset



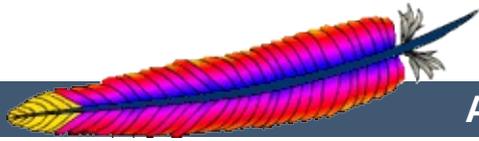
Using Filters

There are two ways to use filtering: Simple and Dynamic. In general, you should use one or the other; mixing them can have unexpected consequences (although simple Input filtering can be mixed freely with either simple or dynamic Output filtering).

The Simple Way is the only way to configure input filters, and is sufficient for output filters where you need a static filter chain. Relevant directives are [SetInputFilter](#), [SetOutputFilter](#), [AddInputFilter](#), [AddOutputFilter](#), [RemoveInputFilter](#), and [RemoveOutputFilter](#).

The Dynamic Way enables both static and flexible, dynamic configuration of output filters, as discussed in the [mod_filter](#) page. Relevant directives are [FilterChain](#), [FilterDeclare](#), and [FilterProvider](#).

One further directive [AddOutputFilterByType](#) is still supported, but may be problematic and is now deprecated. Use dynamic configuration instead.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

suEXEC Support

The **suEXEC** feature provides Apache users the ability to run **CGI** and **SSI** programs under user IDs different from the user ID of the calling web server. Normally, when a CGI or SSI program executes, it runs as the same user who is running the web server.

Used properly, this feature can reduce considerably the security risks involved with allowing users to develop and run private CGI or SSI programs. However, if suEXEC is improperly configured, it can cause any number of problems and possibly create new holes in your computer's security. If you aren't familiar with managing *setuid root* programs and the security issues they present, we highly recommend that you not consider using suEXEC.



Before jumping head-first into this document, you should be aware of the assumptions made on the part of the Apache Group and this document.

First, it is assumed that you are using a UNIX derivative operating system that is capable of **setuid** and **setgid** operations. All command examples are given in this regard. Other platforms, if they are capable of supporting suEXEC, may differ in their configuration.

Second, it is assumed you are familiar with some basic concepts of your computer's security and its administration. This involves an understanding of **setuid/setgid** operations and the various effects they may have on your system and its level of security.

Third, it is assumed that you are using an **unmodified** version of suEXEC code. All code for suEXEC has been carefully scrutinized and tested by the developers as well as numerous beta testers. Every precaution has been taken to ensure a simple yet solidly safe base of code. Altering this code can cause unexpected problems and new security risks. It is **highly** recommended you not alter the suEXEC code unless you are well versed in the particulars of security programming and are willing to share your work with the Apache Group for consideration.

Fourth, and last, it has been the decision of the Apache Group to **NOT** make suEXEC part of the default installation of Apache. To this end, suEXEC configuration requires of the administrator careful attention to details. After due consideration has been given to the various settings for suEXEC, the administrator may install suEXEC through normal installation methods. The values for these settings need to be carefully determined and specified by the administrator to properly maintain system security during the use of suEXEC functionality. It is through this detailed process that the Apache Group hopes to limit suEXEC installation only to those who are careful and determined enough to use it.

Still with us? Yes? Good. Let's move on!



Security Model

Before we begin configuring and installing suEXEC, we will first discuss the security model you are about to implement. By doing so, you may better understand what exactly is going on inside suEXEC and what precautions are taken to ensure your system's security.

suEXEC is based on a setuid "wrapper" program that is called by the main Apache web server. This wrapper is called when an HTTP request is made for a CGI or SSI program that the administrator has designated to run as a userid other than that of the main server. When such a request is made, Apache provides the suEXEC wrapper with the program's name and the user and group IDs under which the program is to execute.

The wrapper then employs the following process to determine success or failure -- if any one of these conditions fail, the program logs the failure and exits with an error, otherwise it will continue:

- 1. Is the user executing this wrapper a valid user of this system?**

This is to ensure that the user executing the wrapper is truly a user of the system.

- 2. Was the wrapper called with the proper number of arguments?**

The wrapper will only execute if it is given the proper number of arguments. The proper argument format is known to the Apache web server. If the wrapper is not receiving the proper number of arguments, it is either being hacked, or there is something wrong with the suEXEC portion of your Apache binary.

- 3. Is this valid user allowed to run the wrapper?**

Is this user the user allowed to run this wrapper? Only one user (the Apache user) is allowed to execute this program.

4. Does the target CGI or SSI program have an unsafe hierarchical reference?

Does the target CGI or SSI program's path contain a leading '/' or have a '..' backreference? These are not allowed; the target CGI/SSI program must reside within suEXEC's document root (see `--with-suexec-docroot=DIR` below).

5. Is the target user name valid?

Does the target user exist?

6. Is the target group name valid?

Does the target group exist?

7. Is the target user *NOT* superuser?

suEXEC does not allow *root* to execute CGI/SSI programs.

8. Is the target userid *ABOVE* the minimum ID number?

The minimum user ID number is specified during configuration. This allows you to set the lowest possible userid that will be allowed to execute CGI/SSI programs. This is useful to block out "system" accounts.

9. Is the target group *NOT* the superuser group?

Presently, suEXEC does not allow the *root* group to execute CGI/SSI programs.

10. Is the target groupid *ABOVE* the minimum ID number?

The minimum group ID number is specified during configuration. This allows you to set the lowest possible groupid that will be allowed to execute CGI/SSI programs. This is useful to block out

"system" groups.

11. Can the wrapper successfully become the target user and group?

Here is where the program becomes the target user and group via `setuid` and `setgid` calls. The group access list is also initialized with all of the groups of which the user is a member.

12. Can we change directory to the one in which the target CGI/SSI program resides?

If it doesn't exist, it can't very well contain files. If we can't change directory to it, it might aswell not exist.

13. Is the directory within the Apache webospace?

If the request is for a regular portion of the server, is the requested directory within suEXEC's document root? If the request is for a `UserDir`, is the requested directory within the directory configured as suEXEC's userdir (see [suEXEC's configuration options](#))?

14. Is the directory *NOT* writable by anyone else?

We don't want to open up the directory to others; only the owner user may be able to alter this directories contents.

15. Does the target CGI/SSI program exist?

If it doesn't exists, it can't very well be executed.

16. Is the target CGI/SSI program *NOT* writable by anyone else?

We don't want to give anyone other than the owner the ability to change the CGI/SSI program.

17. Is the target CGI/SSI program *NOT* setuid or setgid?

We do not want to execute programs that will then change our UID/GID again.

18. Is the target user/group the same as the program's user/group?

Is the user the owner of the file?

19. Can we successfully clean the process environment to ensure safe operations?

suEXEC cleans the process' environment by establishing a safe execution PATH (defined during configuration), as well as only passing through those variables whose names are listed in the safe environment list (also created during configuration).

20. Can we successfully become the target CGI/SSI program and execute?

Here is where suEXEC ends and the target CGI/SSI program begins.

This is the standard operation of the suEXEC wrapper's security model. It is somewhat stringent and can impose new limitations and guidelines for CGI/SSI design, but it was developed carefully step-by-step with security in mind.

For more information as to how this security model can limit your possibilities in regards to server configuration, as well as what security risks can be avoided with a proper suEXEC setup, see the ["Beware the Jabberwock"](#) section of this document.



Here's where we begin the fun.

suEXEC configuration options

--enable-suexec

This option enables the suEXEC feature which is never installed or activated by default. At least one `--with-suexec-xxxxx` option has to be provided together with the `--enable-suexec` option to let APACI accept your request for using the suEXEC feature.

--with-suexec-bin=PATH

The path to the suexec binary must be hard-coded in the server for security reasons. Use this option to override the default path. e.g. `--with-suexec-bin=/usr/sbin/suexec`

--with-suexec-caller=UID

The [username](#) under which Apache normally runs. This is the only user allowed to execute this program.

--with-suexec-userdir=DIR

Define to be the subdirectory under users' home directories where suEXEC access should be allowed. All executables under this directory will be executable by suEXEC as the user so they should be "safe" programs. If you are using a "simple" [UserDir](#) directive (ie. one without a "*" in it) this should be set to the same value. suEXEC will not work properly in cases where the [UserDir](#) directive points to a location that is not the same as the user's home directory as referenced in the passwd file. Default value is "public_html". If you have virtual hosts with a different [UserDir](#) for each, you will need to define them to all reside in one parent directory; then name that parent directory here. **If this is not defined properly, "~userdir" cgi requests will not work!**

--with-suexec-docroot=DIR

Define as the DocumentRoot set for Apache. This will be the only

hierarchy (aside from [UserDirs](#)) that can be used for suEXEC behavior. The default directory is the `--datadir` value with the suffix `/htdocs`, e.g. if you configure with `--datadir=/home/apache` the directory `/home/apache/htdocs` is used as document root for the suEXEC wrapper.

`--with-suexec-uidmin=UID`

Define this as the lowest UID allowed to be a target user for suEXEC. For most systems, 500 or 100 is common. Default value is 100.

`--with-suexec-gidmin=GID`

Define this as the lowest GID allowed to be a target group for suEXEC. For most systems, 100 is common and therefore used as default value.

`--with-suexec-logfile=FILE`

This defines the filename to which all suEXEC transactions and errors are logged (useful for auditing and debugging purposes). By default the logfile is named `suexec_log` and located in your standard logfile directory (`--logfiledir`).

`--with-suexec-safepath=PATH`

Define a safe PATH environment to pass to CGI executables. Default value is `/usr/local/bin:/usr/bin:/bin`.

Compiling and installing the suEXEC wrapper

If you have enabled the suEXEC feature with the `--enable-suexec` option the suexec binary (together with Apache itself) is automatically built if you execute the make command.

After all components have been built you can execute the command `make install` to install them. The binary image suexec is installed in the directory defined by the `--sbindir` option. The default location is

"/usr/local/apache2/bin/suexec".

Please note that you need **root privileges** for the installation step. In order for the wrapper to set the user ID, it must be installed as owner *root* and must have the setuserid execution bit set for file modes.

Setting paranoid permissions

Although the suEXEC wrapper will check to ensure that its caller is the correct user as specified with the `--with-suexec-caller` [configure](#) option, there is always the possibility that a system or library call suEXEC uses before this check may be exploitable on your system. To counter this, and because it is best-practise in general, you should use filesystem permissions to ensure that only the group Apache runs as may execute suEXEC.

If for example, your web server is configured to run as:

```
User www
Group webgroup
```

and [suexec](#) is installed at "/usr/local/apache2/bin/suexec", you should run:

```
chgrp webgroup /usr/local/apache2/bin/suexec
chmod 4750 /usr/local/apache2/bin/suexec
```

This will ensure that only the group Apache runs as can even execute the suEXEC wrapper.



Upon startup of Apache, it looks for the file [suexec](#) in the directory defined by the `--sbindir` option (default is `"/usr/local/apache/sbin/suexec"`). If Apache finds a properly configured suEXEC wrapper, it will print the following message to the error log:

```
[notice] suEXEC mechanism enabled (wrapper: /path/to/suexec)
```

If you don't see this message at server startup, the server is most likely not finding the wrapper program where it expects it, or the executable is not installed *setuid root*.

If you want to enable the suEXEC mechanism for the first time and an Apache server is already running you must kill and restart Apache. Restarting it with a simple HUP or USR1 signal will not be enough.

If you want to disable suEXEC you should kill and restart Apache after you have removed the [suexec](#) file.



Requests for CGI programs will call the suEXEC wrapper only if they are for a virtual host containing a [SuexecUserGroup](#) directive or if they are processed by [mod_userdir](#).

Virtual Hosts:

One way to use the suEXEC wrapper is through the [SuexecUserGroup](#) directive in [VirtualHost](#) definitions. By setting this directive to values different from the main server user ID, all requests for CGI resources will be executed as the *User* and *Group* defined for that [<VirtualHost>](#). If this directive is not specified for a [<VirtualHost>](#) then the main server userid is assumed.

User directories:

Requests that are processed by [mod_userdir](#) will call the suEXEC wrapper to execute CGI programs under the userid of the requested user directory. The only requirement needed for this feature to work is for CGI execution to be enabled for the user and that the script must meet the scrutiny of the [security checks](#) above. See also the `--with-suexec-userdir` [compile time option](#).



Logging output

The suEXEC wrapper will write log information to the file defined with the `--with-suexec-logfile` option as indicated above. If you feel you have configured and installed the wrapper properly, have a look at this log and the `error_log` for the server to see where you may have gone astray.



NOTE! This section may not be complete. For the latest revision of this section of the documentation, see the Apache Group's [Online Documentation](#) version.

There are a few points of interest regarding the wrapper that can cause limitations on server setup. Please review these before submitting any "bugs" regarding suEXEC.

- **suEXEC Points Of Interest**
- Hierarchy limitations

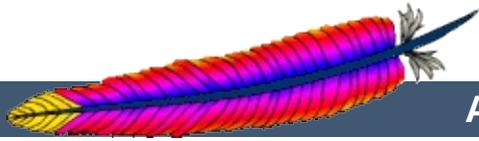
For security and efficiency reasons, all suEXEC requests must remain within either a top-level document root for virtual host requests, or one top-level personal document root for userdir requests. For example, if you have four VirtualHosts configured, you would need to structure all of your VHosts' document roots off of one main Apache document hierarchy to take advantage of suEXEC for VirtualHosts. (Example forthcoming.)

- suEXEC's PATH environment variable

This can be a dangerous thing to change. Make certain every path you include in this define is a **trusted** directory. You don't want to open people up to having someone from across the world running a trojan horse on them.

- Altering the suEXEC code

Again, this can cause **Big Trouble** if you try this without knowing what you are doing. Stay away from it if at all possible.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Issues Regarding DNS and Apache

This page could be summarized with the statement: don't configure Apache in such a way that it relies on DNS resolution for parsing of the configuration files. If Apache requires DNS resolution to parse the configuration files then your server may be subject to reliability problems (ie. it might not boot), or denial and theft of service attacks (including users able to steal hits from other users).



```
<VirtualHost www.abc.dom>
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
</VirtualHost>
```

In order for Apache to function properly, it absolutely needs to have two pieces of information about each virtual host: the [ServerName](#) and at least one IP address that the server will bind and respond to. The above example does not include the IP address, so Apache must use DNS to find the address of `www.abc.dom`. If for some reason DNS is not available at the time your server is parsing its config file, then this virtual host **will not be configured**. It won't be able to respond to any hits to this virtual host (prior to Apache version 1.2 the server would not even boot).

Suppose that `www.abc.dom` has address `192.0.2.1`. Then consider this configuration snippet:

```
<VirtualHost 192.0.2.1>
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
</VirtualHost>
```

This time Apache needs to use reverse DNS to find the `ServerName` for this virtualhost. If that reverse lookup fails then it will partially disable the virtualhost (prior to Apache version 1.2 the server would not even boot). If the virtual host is name-based then it will effectively be totally disabled, but if it is IP-based then it will mostly work. However, if Apache should ever have to generate a full URL for the server which includes the server name, then it will fail to generate a valid URL.

Here is a snippet that avoids both of these problems:

```
<VirtualHost 192.0.2.1>
ServerName www.abc.dom
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
```

```
</VirtualHost>
```



There are (at least) two forms that denial of service can come in. If you are running a version of Apache prior to version 1.2 then your server will not even boot if one of the two DNS lookups mentioned above fails for any of your virtual hosts. In some cases this DNS lookup may not even be under your control; for example, if `abc . dom` is one of your customers and they control their own DNS, they can force your (pre-1.2) server to fail while booting simply by deleting the `www . abc . dom` record.

Another form is far more insidious. Consider this configuration snippet:

```
<VirtualHost www.abc.dom>
  ServerAdmin webgirl@abc.dom
  DocumentRoot /www/abc
</VirtualHost>

<VirtualHost www.def.dom>
  ServerAdmin webguy@def.dom
  DocumentRoot /www/def
</VirtualHost>
```

Suppose that you've assigned `192.0.2.1` to `www . abc . dom` and `192.0.2.2` to `www . def . dom`. Furthermore, suppose that `abc . dom` has control of their own DNS. With this config you have put `abc . dom` into a position where they can steal all traffic destined to `def . dom`. To do so, all they have to do is set `www . abc . dom` to `192.0.2.2`. Since they control their own DNS you can't stop them from pointing the `www . abc . dom` record wherever they wish.

Requests coming in to `192.0.2.2` (including all those where users typed in URLs of the form `http://www . def . dom/whatever`) will all be served by the `abc . dom` virtual host. To better understand why this happens requires a more in-depth discussion of how Apache matches up incoming requests with the virtual host that will serve it. A rough document describing this [is available](#).



THE MAIN SERVER ADDRESS

The addition of [name-based virtual host support](#) in Apache 1.1 requires Apache to know the IP address(es) of the host that [httpd](#) is running on. To get this address it uses either the global [ServerName](#) (if present) or calls the C function `gethostname` (which should return the same as typing "hostname" at the command prompt). Then it performs a DNS lookup on this address. At present there is no way to avoid this lookup.

If you fear that this lookup might fail because your DNS server is down then you can insert the hostname in `/etc/hosts` (where you probably already have it so that the machine can boot properly). Then ensure that your machine is configured to use `/etc/hosts` in the event that DNS fails. Depending on what OS you are using this might be accomplished by editing `/etc/resolv.conf`, or maybe `/etc/nsswitch.conf`.

If your server doesn't have to perform DNS for any other reason then you might be able to get away with running Apache with the `HOSTRESORDER` environment variable set to "local". This all depends on what OS and resolver libraries you are using. It also affects CGIs unless you use [mod_env](#) to control the environment. It's best to consult the man pages or FAQs for your OS.



-
- use IP addresses in VirtualHost
 - use IP addresses in Listen
 - ensure all virtual hosts have an explicit ServerName
 - create a `<VirtualHost _default_*>` server that has no pages to serve



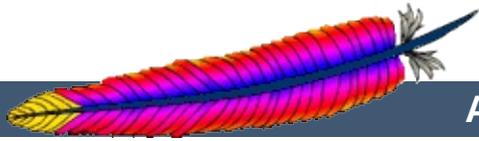
Appendix: DNS Problems

The situation regarding DNS is highly undesirable. For Apache 1.2 we've attempted to make the server at least continue booting in the event of failed DNS, but it might not be the best we can do. In any event, requiring the use of explicit IP addresses in configuration files is highly undesirable in today's Internet where renumbering is a necessity.

A possible work around to the theft of service attack described above would be to perform a reverse DNS lookup on the IP address returned by the forward lookup and compare the two names -- in the event of a mismatch, the virtualhost would be disabled. This would require reverse DNS to be configured properly (which is something that most admins are familiar with because of the common use of "double-reverse" DNS lookups by FTP servers and TCP wrappers).

In any event, it doesn't seem possible to reliably boot a virtual-hosted web server when DNS has failed unless IP addresses are used. Partial solutions such as disabling portions of the configuration might be worse than not booting at all depending on what the webserver is supposed to accomplish.

As HTTP/1.1 is deployed and browsers and proxies start issuing the Host header it will become possible to avoid the use of IP-based virtual hosts entirely. In this case, a webserver has no requirement to do DNS lookups during configuration. But as of March 1997 these features have not been deployed widely enough to be put into use on critical web servers.



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Virtual Hosts](#)

Unterstützung namensbasierter virtueller Hosts

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Das Dokument beschreibt, wann und wie namensbasierte virtuelle Hosts zu verwenden sind.

Siehe auch

[Unterstützung IP-basierter virtueller Hosts](#)

[Tiefergehende Erörterung der Zuweisung virtueller Hosts](#)

[Dynamisch konfiguriertes Massen-Virtual-Hosting](#)

[Beispiele für virtuelle Hosts in typischen Installationen](#)

[ServerPath-Beispielkonfiguration](#)



IP-basierte virtuelle Hosts verwenden die IP-Adresse der Verbindung, um den korrekten virtuellen Host zur Bedienung einer Anfrage zu ermitteln. Folglich benötigen Sie eine IP-Adresse für jeden virtuellen Host. Bei der Verwendung von namensbasierten virtuellen Hosts verläßt sich der Server darauf, dass der Client den Hostnamen als Bestandteil der HTTP-Header angibt. Durch Anwendung dieser Technik können sich mehrere verschiedene Hosts die gleiche IP-Adresse teilen.

Die Verwendung von namensbasierten virtuellen Hosts ist gewöhnlich einfacher. Sie müssen lediglich Ihren DNS-Server darauf einstellen, jeden Hostnamen auf die richtige IP-Adresse abzubilden, und dann den Apache HTTP Server so konfigurieren, dass er die verschiedenen Hostnamen erkennt. Namensbasierte virtuelle Hosts entschärfen auch den Bedarf an knappen IP-Adressen. Daher sollten Sie namensbasierte virtuelle Hosts verwenden, sofern kein besonderer Grund dafür existiert, IP-basierte virtuelle Hosts zu wählen. Mögliche Gründe für die Verwendung IP-basierter virtueller Hosts sind:

- Einige antike Clients sind nicht kompatibel zu namensbasierten virtuellen Hosts. Damit namensbasierte virtuelle Hosts funktionieren, muss der Client den HTTP-Host-Header senden. Dies ist bei HTTP/1.1 vorgeschrieben und in allen modernen HTTP/1.0-Browsern als Erweiterung implementiert. Wenn Sie Unterstützung für veraltete Clients benötigen und dennoch namensbasierte virtuelle Hosts verwenden, dann finden Sie eine mögliche Lösung dafür am Ende des Dokuments.
- Namensbasierte virtuelle Hosts können aufgrund der Natur des SSL-Protokolls nicht mit SSL-gesicherten Servern verwendet werden.
- Einige Betriebssysteme und Netzwerkanlagen setzen Techniken zum Bandbreiten-Management ein, die nicht zwischen Hosts unterscheiden können, wenn diese nicht auf verschiedenen IP-Adressen liegen.



Referenzierte Module	Referenzierte Direktiven
----------------------	--------------------------

core

DocumentRoot

NameVirtualHost

ServerAlias

ServerName

ServerPath

<VirtualHost>

Um namensbasierte virtuelle Hosts zu verwenden, müssen Sie die IP-Adresse (und möglicherweise den Port) des Servers benennen, an der Anfragen für die Hosts entgegengenommen werden. Dies wird mit der Direktive NameVirtualHost eingestellt. Im Normalfall, wenn alle IP-Adressen des Server verwendet werden sollen, können Sie * als Argument für NameVirtualHost verwenden. Wenn Sie vorhaben, mehrere Ports zu nutzen (etwa wenn SSL läuft), sollten Sie dem Argument einen Port hinzufügen, wie zum Beispiel *:80. Beachten Sie, dass die Angabe einer IP-Adresse in einer NameVirtualHost-Anweisung den Server nicht automatisch an dieser Adresse lauschen lässt. Lesen Sie bitte "[Bestimmen der vom Apache verwendeten Adressen und Ports](#)" für weitere Details. Zusätzlich muss jede hier angegebene IP-Adresse einer Netzwerkkarte des Servers zugeordnet sein.

Der nächste Schritt ist die Erstellung eines <VirtualHost>-Blocks für jeden einzelnen Host, den Sie bedienen wollen. Das Argument der Direktive <VirtualHost> sollte das gleiche sein wie das Argument der NameVirtualHost-Anweisung (d.h. eine IP-Adresse oder * für alle Adressen). Innerhalb jedes <VirtualHost>-Blocks benötigen Sie zumindestens eine ServerName-Anweisung, um zu bestimmen, welcher Host bedient wird, und eine DocumentRoot-Anweisung, um anzugeben, wo im Dateisystem der Inhalt des Hosts abgelegt ist.

Der Hauptserver verschwindet

Wenn Sie virtuelle Hosts zu einem bestehenden Webserver hinzufügen, müssen Sie auch einen `<VirtualHost>`-Block für den bestehenden Host (*Anm.d.Ü.:* und bisherigen Hauptserver) erstellen. Die `ServerName`- und `DocumentRoot`-Anweisungen zu diesem virtuellen Host sollten die gleichen sein wie die globalen `ServerName`- und `DocumentRoot`-Anweisungen. Führen Sie diesen virtuellen Host als erstes in der Konfigurationsdatei auf, so dass er als Standard-Host fungiert.

Vorausgesetzt, Sie bedienen z.B. die Domain `www.domain.tld` und möchten den virtuellen Host `www.otherdomain.tld` hinzufügen, welcher auf die gleiche IP-Adresse zeigt. Dann fügen Sie einfach Folgendes der `httpd.conf` hinzu:

```
NameVirtualHost *:80

<VirtualHost *:80>
    ServerName www.domain.tld
    ServerAlias domain.tld *.domain.tld
    DocumentRoot /www/domain
</VirtualHost>

<VirtualHost *:80>
    ServerName www.otherdomain.tld
    DocumentRoot /www/otherdomain
</VirtualHost>
```

Sie können anstelle des `*` bei den beiden Anweisungen `NameVirtualHost` und `<VirtualHost>` alternativ eine eindeutige IP-Adresse angeben. Das kann man beispielsweise machen, um einige namensbasierte virtuelle Hosts auf einer IP-Adresse zu betreiben und entweder IP-basierte oder ein anderes Set von namensbasierten virtuellen Hosts auf einer anderen Adresse.

Viele Server wollen unter mehr als einem Namen erreichbar sein. Die Direktive `ServerAlias`, die innerhalb des `<VirtualHost>`-

Abschnittes angegeben wird, ermöglicht dies. Zum Beispiel zeigt die [ServerAlias](#)-Anweisung in dem ersten [<VirtualHost>](#)-Block oben an, dass die aufgeführten Namen alternative Namen sind, die man verwenden kann, um das gleiche Webangebot zu erreichen:

```
ServerAlias domain.tld *.domain.tld
```

Anfragen für alle Hosts der Domain `domain.tld` werden von dem virtuellen Host `www.domain.tld` bedient. Die Platzhalter `*` und `?` können anstelle entsprechender Namen verwendet werden. Natürlich können Sie nicht einfach Namen erfinden und diese bei [ServerName](#) oder [ServerAlias](#) angeben, Sie müssen zunächst Ihren DNS Server entsprechend konfigurieren, dass er diese Namen auf die mit Ihrem Server verknüpfte IP-Adresse abbildet.

Und schließlich können Sie die Konfiguration der virtuellen Hosts mittels Angabe weiterer Direktiven innerhalb der [<VirtualHost>](#)-Container feineinstellen. Die meisten Direktiven können in diesen Containern angegeben werden und verändern dann ausschließlich die Konfiguration des entsprechenden virtuellen Hosts. Prüfen Sie den [Kontext](#) einer Direktive, um herauszufinden, ob eine bestimmte Direktive zulässig ist. Im *Hauptserver-Kontext* (außerhalb der [<VirtualHost>](#)-Container) definierte Konfigurationsanweisungen werden nur dann angewendet, wenn sie nicht durch Einstellungen des virtuellen Hosts außer Kraft gesetzt wurden.

Wenn nun eine Anfrage eintrifft, prüft der Server zuerst, ob sie eine IP-Adresse verwendet, die der [NameVirtualHost](#)-Anweisung entspricht. Ist dies der Fall, dann sieht er sich jeden [<VirtualHost>](#)-Abschnitt mit einer passenden IP-Adresse an und versucht den einen zu finden, dessen [ServerName](#)- oder [ServerAlias](#)-Anweisung mit dem gewünschten Hostnamen übereinstimmt. Findet er einen, dann verwendet er die Konfiguration dieses Servers. Wird kein passender virtueller Host gefunden, dann wird **der erste angegeben virtuelle Host**

verwendet, dessen IP-Adresse paßt.

Die Folge davon ist, dass der erste aufgeführte virtuelle Host der *Standard-Virtual-Host* ist. Die DocumentRoot-Anweisung des *Hauptservers* wird **niemals** verwendet, wenn eine IP-Adresse mit einer NameVirtualHost-Anweisung übereinstimmt. Wenn Sie eine spezielle Konfiguration für Anfragen angeben möchten, die keinem bestimmten virtuellen Host entsprechen, packen Sie diese Konfiguration einfach in einen <VirtualHost>-Container und führen diesen als erstes in der Konfigurationsdatei auf.



Wie zuvor erwähnt gibt es einige Clients, die nicht die notwendigen Daten senden, mit denen namensbasierte virtuelle Hosts korrekt funktionieren. Diesen Clients werden stets die Seiten des ersten, für diese IP-Adresse aufgeführten virtuellen Hosts gesendet werden (des *primären* namensbasierten virtuellen Hosts).

Was bedeutet älter?

Beachten Sie bitte, wenn wir von älter sprechen, meinen wir auch älter. Es ist sehr unwahrscheinlich, dass sie einen dieser Browser heutzutage in Verwendung finden werden. Alle aktuellen Browser-Versionen senden den Host-Header, so wie er für namensbasierte virtuelle Hosts benötigt wird.

Mit der Direktive `ServerPath` existiert eine mögliche Behelfskonstruktion, obgleich sie etwas schwerfällig ist:

Beispielkonfiguration:

```
NameVirtualHost 111.22.33.44

<VirtualHost 111.22.33.44>
  ServerName www.domain.tld
  ServerPath /domain
  DocumentRoot /web/domain
</VirtualHost>
```

Was bedeutet das? Es bedeutet, dass eine Anfrage für eine mit `"/domain"` beginnende URI von dem virtuellen Host `www.domain.tld` bedient wird. Dies heißt, dass die Seiten für alle Clients unter `http://www.domain.tld/domain/` abrufbar sind, wenngleich Clients, die den Header `Host:` senden, auch über `http://www.domain.tld/` auf sie zugreifen können.

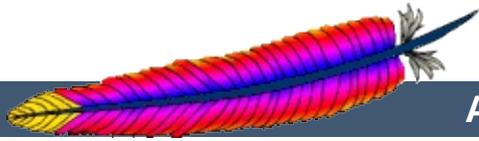
Legen Sie einen Link auf der Seite Ihres primären virtuellen Hosts zu

`http://www.domain.tld/domain/`, um die Behelfslösung verfügbar zu machen. Bei den Seiten der virtuellen Hosts müssen Sie dann sicherstellen, entweder ausschließlich relative Links (z.B. `file.html` oder `../icons/image.gif`) zu verwenden oder Links, die das einleitende `/domain/` enthalten (z.B., `http://www.domain.tld/domain/misc/file.html` oder `/domain/misc/file.html`).

Dies erfordert etwas Disziplin, die Befolgung dieser Richtlinien stellt jedoch größtenteils sicher, dass Ihre Seiten mit allen Browsern funktionieren, alten wie neuen.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Virtual Hosts](#)

Apache IP-based Virtual Host Support

See also

[Name-based Virtual Hosts Support](#)



IP-based virtual hosting

IP-based virtual hosting is a method to apply different directives based on the IP address and port a request is received on. Most commonly, this is used to serve different websites on different ports or interfaces.

In many cases, [name-based virtual hosts](#) are more convenient, because they allow many virtual hosts to share a single address/port. See [Name-based vs. IP-based Virtual Hosts](#) to help you decide.



As the term *IP-based* indicates, the server **must have a different IP address/port combination for each IP-based virtual host**. This can be achieved by the machine having several physical network connections, or by use of virtual interfaces which are supported by most modern operating systems (see system documentation for details, these are frequently called "ip aliases", and the "ifconfig" command is most commonly used to set them up), and/or using multiple port numbers.

In the terminology of Apache HTTP Server, using a single IP address but multiple TCP ports, is also IP-based virtual hosting.



How to Setup Apache

There are two ways of configuring apache to support multiple hosts. Either by running a separate [httpd](#) daemon for each hostname, or by running a single daemon which supports all the virtual hosts.

Use multiple daemons when:

- There are security partitioning issues, such as company1 does not want anyone at company2 to be able to read their data except via the web. In this case you would need two daemons, each running with different [User](#), [Group](#), [Listen](#), and [ServerRoot](#) settings.
- You can afford the memory and file descriptor requirements of listening to every IP alias on the machine. It's only possible to [Listen](#) to the "wildcard" address, or to specific addresses. So if you have a need to listen to a specific address for whatever reason, then you will need to listen to all specific addresses. (Although one [httpd](#) could listen to N-1 of the addresses, and another could listen to the remaining address.)

Use a single daemon when:

- Sharing of the httpd configuration between virtual hosts is acceptable.
- The machine services a large number of requests, and so the performance loss in running separate daemons may be significant.



Setting up multiple instances

Create a separate [httpd](#) installation for each virtual host. For each installation, use the [Listen](#) directive in the configuration file to select which IP address (or virtual host) that daemon services. e.g.

```
Listen 192.168.0.1:80
```

It is recommended that you use an IP address instead of a hostname (see [DNS caveats](#)).



Setting up a single server with virtual hosts

For this case, a single [httpd](#) will service requests for the main server and all the virtual hosts. The [VirtualHost](#) directive in the configuration file is used to set the values of [ServerAdmin](#), [ServerName](#), [DocumentRoot](#), [ErrorLog](#) and [TransferLog](#) or [CustomLog](#) configuration directives to different values for each virtual host. e.g.

```
<VirtualHost 192.168.0.1:80>
ServerAdmin webmaster@smallco.example.com
DocumentRoot /groups/smallco/www
ServerName smallco.example.com
ErrorLog /groups/smallco/logs/error_log
TransferLog /groups/smallco/logs/access_log
</VirtualHost>

<VirtualHost 192.168.0.2:80>
ServerAdmin webmaster@baygroup.example.org
DocumentRoot /groups/baygroup/www
ServerName baygroup.example.com
ErrorLog /groups/baygroup/logs/error_log
TransferLog /groups/baygroup/logs/access_log
</VirtualHost>
```

It is recommended that you use an IP address instead of a hostname in the `<VirtualHost>` directive (see [DNS caveats](#)).

Specific IP addresses or ports have precedence over their wildcard equivalents, and any virtual host that matches has precedence over the servers base configuration.

Almost **any** configuration directive can be put in the `VirtualHost` directive, with the exception of directives that control process creation and a few other directives. To find out if a directive can be used in the `VirtualHost` directive, check the [Context](#) using the [directive index](#).

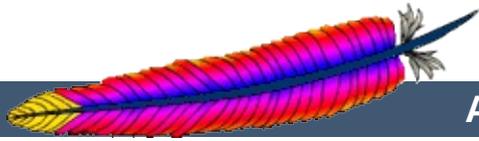
[SuexecUserGroup](#) may be used inside a `VirtualHost` directive if the [suEXEC wrapper](#) is used.

SECURITY: When specifying where to write log files, be aware of some

security risks which are present if anyone other than the user that starts Apache has write access to the directory where they are written. See the [security tips](#) document for details.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Virtual Hosts](#)

Dynamically configured mass virtual hosting

This document describes how to efficiently serve an arbitrary number of virtual hosts with Apache.



The techniques described here are of interest if your `httpd.conf` contains many `<VirtualHost>` sections that are substantially the same, for example:

```
NameVirtualHost 111.22.33.44
<VirtualHost 111.22.33.44>
    ServerName www.customer-1.com
    DocumentRoot /www/hosts/www.customer-1.com/docs
    ScriptAlias /cgi-bin/ /www/hosts/www.customer-1.com/cgi-bin
</VirtualHost>
<VirtualHost 111.22.33.44>
    ServerName www.customer-2.com
    DocumentRoot /www/hosts/www.customer-2.com/docs
    ScriptAlias /cgi-bin/ /www/hosts/www.customer-2.com/cgi-bin
</VirtualHost>
# blah blah blah
<VirtualHost 111.22.33.44>
    ServerName www.customer-N.com
    DocumentRoot /www/hosts/www.customer-N.com/docs
    ScriptAlias /cgi-bin/ /www/hosts/www.customer-N.com/cgi-bin
</VirtualHost>
```

The basic idea is to replace all of the static `<VirtualHost>` configuration with a mechanism that works it out dynamically. This has a number of advantages:

1. Your configuration file is smaller so Apache starts faster and uses less memory.
2. Adding virtual hosts is simply a matter of creating the appropriate directories in the filesystem and entries in the DNS - you don't need to reconfigure or restart Apache.

The main disadvantage is that you cannot have a different log file for each virtual host; however if you have very many virtual hosts then doing this is dubious anyway because it eats file descriptors. It is better to log to a pipe or a fifo and arrange for the process at the other end to distribute the logs to the customers (it can also accumulate statistics, etc.).



A virtual host is defined by two pieces of information: its IP address, and the contents of the Host : header in the HTTP request. The dynamic mass virtual hosting technique is based on automatically inserting this information into the pathname of the file that is used to satisfy the request. This is done most easily using [mod_vhost_alias](#), but if you are using a version of Apache up to 1.3.6 then you must use [mod_rewrite](#). Both of these modules are disabled by default; you must enable one of them when configuring and building Apache if you want to use this technique.

A couple of things need to be `faked' to make the dynamic virtual host look like a normal one. The most important is the server name which is used by Apache to generate self-referential URLs, etc. It is configured with the ServerName directive, and it is available to CGIs via the SERVER_NAME environment variable. The actual value used at run time is controlled by the [UseCanonicalName](#) setting. With UseCanonicalName Off the server name comes from the contents of the Host : header in the request. With UseCanonicalName DNS it comes from a reverse DNS lookup of the virtual host's IP address. The former setting is used for name-based dynamic virtual hosting, and the latter is used for IP-based hosting. If Apache cannot work out the server name because there is no Host : header or the DNS lookup fails then the value configured with ServerName is used instead.

The other thing to `fake' is the document root (configured with DocumentRoot and available to CGIs via the DOCUMENT_ROOT environment variable). In a normal configuration this setting is used by the core module when mapping URIs to filenames, but when the server is configured to do dynamic virtual hosting that job is taken over by another module (either [mod_vhost_alias](#) or [mod_rewrite](#)) which has a different way of doing the mapping. Neither of these modules is responsible for setting the DOCUMENT_ROOT environment variable so if any CGIs or SSI documents make use of it they will get a misleading

value.



This extract from `httpd.conf` implements the virtual host arrangement outlined in the [Motivation](#) section above, but in a generic fashion using `mod_vhost_alias`.

```
# get the server name from the Host: header
UseCanonicalName Off

# this log format can be split per-virtual-host based on the first
field
LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# include the server name in the filenames used to satisfy requests
VirtualDocumentRoot /www/hosts/%0/docs
VirtualScriptAlias /www/hosts/%0/cgi-bin
```

This configuration can be changed into an IP-based virtual hosting solution by just turning `UseCanonicalName Off` into `UseCanonicalName DNS`. The server name that is inserted into the filename is then derived from the IP address of the virtual host.



This is an adjustment of the above system tailored for an ISP's homepages server. Using a slightly more complicated configuration we can select substrings of the server name to use in the filename so that e.g. the documents for `www.user.isp.com` are found in `/home/user/`. It uses a single `cgi-bin` directory instead of one per virtual host.

```
# all the preliminary stuff is the same as above, then

# include part of the server name in the filenames
VirtualDocumentRoot /www/hosts/%2/docs

# single cgi-bin directory
ScriptAlias /cgi-bin/ /www/std-cgi/
```

There are examples of more complicated `VirtualDocumentRoot` settings in the [mod_vhost_alias](#) documentation.



server

With more complicated setups you can use Apache's normal `<VirtualHost>` directives to control the scope of the various virtual hosting configurations. For example, you could have one IP address for homepages customers and another for commercial customers with the following setup. This can of course be combined with conventional `<VirtualHost>` configuration sections.

```
UseCanonicalName Off

LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon

<Directory /www/commercial>
  Options FollowSymLinks
  AllowOverride All
</Directory>

<Directory /www/homepages>
  Options FollowSymLinks
  AllowOverride None
</Directory>

<VirtualHost 111.22.33.44>
  ServerName www.commercial.isp.com

  CustomLog logs/access_log.commercial vcommon

  VirtualDocumentRoot /www/commercial/%0/docs
  VirtualScriptAlias /www/commercial/%0/cgi-bin
</VirtualHost>

<VirtualHost 111.22.33.45>
  ServerName www.homepages.isp.com

  CustomLog logs/access_log.homepages vcommon

  VirtualDocumentRoot /www/homepages/%0/docs
  ScriptAlias /cgi-bin/ /www/std-cgi/
</VirtualHost>
```

Note

If the first VirtualHost block does *not* include a ServerName directive, the reverse DNS of the relevant IP will be used instead. If this is not the server name you wish to use, a bogus entry (ServerName none.example.com) can be added to get around this behaviour.



After [the first example](#) I noted that it is easy to turn it into an IP-based virtual hosting setup. Unfortunately that configuration is not very efficient because it requires a DNS lookup for every request. This can be avoided by laying out the filesystem according to the IP addresses themselves rather than the corresponding names and changing the logging similarly. Apache will then usually not need to work out the server name and so incur a DNS lookup.

```
# get the server name from the reverse DNS of the IP address
UseCanonicalName DNS

# include the IP address in the logs so they may be split
LogFormat "%A %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# include the IP address in the filenames
VirtualDocumentRootIP /www/hosts/%0/docs
VirtualScriptAliasIP /www/hosts/%0/cgi-bin
```



This extract from `httpd.conf` does the same thing as [the first example](#). The first half is very similar to the corresponding part above but with some changes for backward compatibility and to make the `mod_rewrite` part work properly; the second half configures `mod_rewrite` to do the actual work.

There are a couple of especially tricky bits: By default, `mod_rewrite` runs before the other URI translation modules (`mod_alias` etc.) so if they are used then `mod_rewrite` must be configured to accommodate them. Also, some magic must be performed to do a per-dynamic-virtual-host equivalent of `ScriptAlias`.

```
# get the server name from the Host: header
UseCanonicalName Off

# splittable logs
LogFormat "%{Host}i %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

<Directory /www/hosts>
  # ExecCGI is needed here because we can't force
  # CGI execution in the way that ScriptAlias does
  Options FollowSymLinks ExecCGI
</Directory>

# now for the hard bit

RewriteEngine On

# a ServerName derived from a Host: header may be any case at all
RewriteMap lowercase int:tolower

## deal with normal documents first:
# allow Alias /icons/ to work - repeat for other aliases
RewriteCond %{REQUEST_URI} !^/icons/
# allow CGIs to work
RewriteCond %{REQUEST_URI} !^/cgi-bin/
# do the magic
RewriteRule ^/(.*)$ /www/hosts/${lowercase:%{SERVER_NAME}}/docs/$1

## and now deal with CGIs - we have to force a MIME type
RewriteCond %{REQUEST_URI} ^/cgi-bin/
```

```
RewriteRule ^/(.*)$ /www/hosts/${lowercase:%{SERVER_NAME}}/cgi-bin/$1  
[T=application/x-httpd-cgi]
```

```
# that's it!
```



This does the same thing as [the second example](#).

```
RewriteEngine on

RewriteMap lowercase int:tolower

# allow CGIs to work
RewriteCond %{REQUEST_URI} !^/cgi-bin/

# check the hostname is right so that the RewriteRule works
RewriteCond ${lowercase:%{SERVER_NAME}} ^www\.[a-z-]+\\.isp\.com$

# concatenate the virtual host name onto the start of the URI
# the [C] means do the next rewrite on the result of this one
RewriteRule ^(.+) ${lowercase:%{SERVER_NAME}}$1 [C]

# now create the real file name
RewriteRule ^www\.[a-z-]+\\.isp\.com/(.*) /home/$1/$2

# define the global CGI directory
ScriptAlias /cgi-bin/ /www/std-cgi/
```



This arrangement uses more advanced `mod_rewrite` features to get the translation from virtual host to document root from a separate configuration file. This provides more flexibility but requires more complicated configuration.

The `vhost.map` file contains something like this:

```
www.customer-1.com /www/customers/1
www.customer-2.com /www/customers/2
# ...
www.customer-N.com /www/customers/N
```

The `http.conf` contains this:

```
RewriteEngine on

RewriteMap lowercase int:tolower

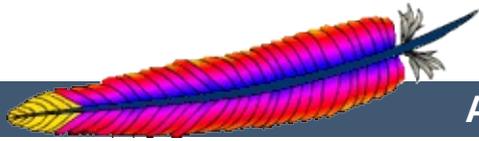
# define the map file
RewriteMap vhost txt:/www/conf/vhost.map

# deal with aliases as above
RewriteCond %{REQUEST_URI} !^/icons/
RewriteCond %{REQUEST_URI} !^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
# this does the file-based remap
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/docs/$1

RewriteCond %{REQUEST_URI} ^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/cgi-bin/$1
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Virtual Hosts](#)

VirtualHost Examples

This document attempts to answer the commonly-asked questions about setting up [virtual hosts](#). These scenarios are those involving multiple web sites running on a single server, via [name-based](#) or [IP-based](#) virtual hosts.



address.

Your server has a single IP address, and multiple aliases (CNAMEs) point to this machine in DNS. You want to run a web server for `www.example.com` and `www.example.org` on this machine.

Note

Creating virtual host configurations on your Apache server does not magically cause DNS entries to be created for those host names. You *must* have the names in DNS, resolving to your IP address, or nobody else will be able to see your web site. You can put entries in your `hosts` file for local testing, but that will work only from the machine with those `hosts` entries.

Server configuration

```
# Ensure that Apache listens on port 80
Listen 80

# Listen for virtual host requests on all IP addresses
NameVirtualHost *:80

<VirtualHost *:80>
    DocumentRoot /www/example1
    ServerName www.example.com

    # Other directives here

</VirtualHost>

<VirtualHost *:80>
    DocumentRoot /www/example2
    ServerName www.example.org

    # Other directives here

</VirtualHost>
```

The asterisks match all addresses, so the main server serves no requests. Due to the fact that `www.example.com` is first in the

configuration file, it has the highest priority and can be seen as the *default* or *primary* server. That means that if a request is received that does not match one of the specified `ServerName` directives, it will be served by this first `VirtualHost`.

Note

You can, if you wish, replace `*` with the actual IP address of the system. In that case, the argument to `VirtualHost` *must* match the argument to `NameVirtualHost`:

```
NameVirtualHost 172.20.30.40  
  
<VirtualHost 172.20.30.40>  
# etc ...
```

However, it is additionally useful to use `*` on systems where the IP address is not predictable - for example if you have a dynamic IP address with your ISP, and you are using some variety of dynamic DNS solution. Since `*` matches any IP address, this configuration would work without changes whenever your IP address changes.

The above configuration is what you will want to use in almost all name-based virtual hosting situations. The only thing that this configuration will not work for, in fact, is when you are serving different content based on differing IP addresses or ports.



Note

Any of the techniques discussed here can be extended to any number of IP addresses.

The server has two IP addresses. On one (172.20.30.40), we will serve the "main" server, server.domain.com and on the other (172.20.30.50), we will serve two or more virtual hosts.

Server configuration

```
Listen 80

# This is the "main" server running on 172.20.30.40
ServerName server.domain.com
DocumentRoot /www/mainserver

# This is the other address
NameVirtualHost 172.20.30.50

<VirtualHost 172.20.30.50>
    DocumentRoot /www/example1
    ServerName www.example.com

    # Other directives here ...
</VirtualHost>

<VirtualHost 172.20.30.50>
    DocumentRoot /www/example2
    ServerName www.example.org

    # Other directives here ...
</VirtualHost>
```

Any request to an address other than 172.20.30.50 will be served from the main server. A request to 172.20.30.50 with an unknown hostname, or no Host : header, will be served from www.example.com.



Returning the same content on different IP addresses (such as an internal and external address).

The server machine has two IP addresses (192.168.1.1 and 172.20.30.40). The machine is sitting between an internal (intranet) network and an external (internet) network. Outside of the network, the name `server.example.com` resolves to the external address (172.20.30.40), but inside the network, that same name resolves to the internal address (192.168.1.1).

The server can be made to respond to internal and external requests with the same content, with just one `VirtualHost` section.

Server configuration

```
NameVirtualHost 192.168.1.1
NameVirtualHost 172.20.30.40

<VirtualHost 192.168.1.1 172.20.30.40>
  DocumentRoot /www/server1
  ServerName server.example.com
  ServerAlias server
</VirtualHost>
```

Now requests from both networks will be served from the same `VirtualHost`.

Note:

On the internal network, one can just use the name `server` rather than the fully qualified host name `server.example.com`.

Note also that, in the above example, you can replace the list of IP addresses with `*`, which will cause the server to respond the same on all addresses.



Running different sites on different ports

You have multiple domains going to the same IP and also want to serve multiple ports. By defining the ports in the "NameVirtualHost" tag, you can allow this to work. If you try using `<VirtualHost name:port>` without the NameVirtualHost name:port or you try to use the Listen directive, your configuration will not work.

Server configuration

```
Listen 80
Listen 8080

NameVirtualHost 172.20.30.40:80
NameVirtualHost 172.20.30.40:8080

<VirtualHost 172.20.30.40:80>
    ServerName www.example.com
    DocumentRoot /www/domain-80
</VirtualHost>

<VirtualHost 172.20.30.40:8080>
    ServerName www.example.com
    DocumentRoot /www/domain-8080
</VirtualHost>

<VirtualHost 172.20.30.40:80>
    ServerName www.example.org
    DocumentRoot /www/otherdomain-80
</VirtualHost>

<VirtualHost 172.20.30.40:8080>
    ServerName www.example.org
    DocumentRoot /www/otherdomain-8080
</VirtualHost>
```



The server has two IP addresses (172.20.30.40 and 172.20.30.50) which resolve to the names `www.example.com` and `www.example.org` respectively.

Server configuration

```
Listen 80
```

```
<VirtualHost 172.20.30.40>  
  DocumentRoot /www/example1  
  ServerName www.example.com  
</VirtualHost>
```

```
<VirtualHost 172.20.30.50>  
  DocumentRoot /www/example2  
  ServerName www.example.org  
</VirtualHost>
```

Requests for any address not specified in one of the `<VirtualHost>` directives (such as `localhost`, for example) will go to the main server, if there is one.



The server machine has two IP addresses (172.20.30.40 and 172.20.30.50) which resolve to the names www.example.com and www.example.org respectively. In each case, we want to run hosts on ports 80 and 8080.

Server configuration

```
Listen 172.20.30.40:80
Listen 172.20.30.40:8080
Listen 172.20.30.50:80
Listen 172.20.30.50:8080

<VirtualHost 172.20.30.40:80>
  DocumentRoot /www/example1-80
  ServerName www.example.com
</VirtualHost>

<VirtualHost 172.20.30.40:8080>
  DocumentRoot /www/example1-8080
  ServerName www.example.com
</VirtualHost>

<VirtualHost 172.20.30.50:80>
  DocumentRoot /www/example2-80
  ServerName www.example.org
</VirtualHost>

<VirtualHost 172.20.30.50:8080>
  DocumentRoot /www/example2-8080
  ServerName www.example.org
</VirtualHost>
```



On some of my addresses, I want to do name-based virtual hosts, and on others, IP-based hosts.

Server configuration

```
Listen 80
```

```
NameVirtualHost 172.20.30.40
```

```
<VirtualHost 172.20.30.40>  
  DocumentRoot /www/example1  
  ServerName www.example.com  
</VirtualHost>
```

```
<VirtualHost 172.20.30.40>  
  DocumentRoot /www/example2  
  ServerName www.example.org  
</VirtualHost>
```

```
<VirtualHost 172.20.30.40>  
  DocumentRoot /www/example3  
  ServerName www.example3.net  
</VirtualHost>
```

```
# IP-based
```

```
<VirtualHost 172.20.30.50>  
  DocumentRoot /www/example4  
  ServerName www.example4.edu  
</VirtualHost>
```

```
<VirtualHost 172.20.30.60>  
  DocumentRoot /www/example5  
  ServerName www.example5.gov  
</VirtualHost>
```



The following example allows a front-end machine to proxy a virtual host through to a server running on another machine. In the example, a virtual host of the same name is configured on a machine at 192.168.111.2. The `ProxyPreserveHost On` directive is used so that the desired hostname is passed through, in case we are proxying multiple hostnames to a single machine.

```
<VirtualHost *:*>
ProxyPreserveHost On
ProxyPass / http://192.168.111.2/
ProxyPassReverse / http://192.168.111.2/
ServerName hostname.example.com
</VirtualHost>
```



`_default_` vhosts for all ports

Catching every request to any unspecified IP address and port, *i.e.*, an address/port combination that is not used for any other virtual host.

Server configuration

```
<VirtualHost _default_:*>
  DocumentRoot /www/default
</VirtualHost>
```

Using such a default vhost with a wildcard port effectively prevents any request going to the main server.

A default vhost never serves a request that was sent to an address/port that is used for name-based vhosts. If the request contained an unknown or no `Host :` header it is always served from the primary name-based vhost (the vhost for that address/port appearing first in the configuration file).

You can use [AliasMatch](#) or [RewriteRule](#) to rewrite any request to a single information page (or script).

`_default_` vhosts for different ports

Same as setup 1, but the server listens on several ports and we want to use a second `_default_` vhost for port 80.

Server configuration

```
<VirtualHost _default_:80>
  DocumentRoot /www/default80
  # ...
</VirtualHost>

<VirtualHost _default_:*>
  DocumentRoot /www/default
  # ...
```

```
</VirtualHost>
```

The default vhost for port 80 (which *must* appear before any default vhost with a wildcard port) catches all requests that were sent to an unspecified IP address. The main server is never used to serve a request.

`_default_` vhosts for one port

We want to have a default vhost for port 80, but no other default vhosts.

Server configuration

```
<VirtualHost _default_:80>  
DocumentRoot /www/default  
...  
</VirtualHost>
```

A request to an unspecified address on port 80 is served from the default vhost. Any other request to an unspecified address and port is served from the main server.



The name-based vhost with the hostname `www.example.org` (from our [name-based](#) example, setup 2) should get its own IP address. To avoid problems with name servers or proxies who cached the old IP address for the name-based vhost we want to provide both variants during a migration phase.

The solution is easy, because we can simply add the new IP address (`172.20.30.50`) to the `VirtualHost` directive.

Server configuration

```
Listen 80
ServerName www.example.com
DocumentRoot /www/example1

NameVirtualHost 172.20.30.40

<VirtualHost 172.20.30.40 172.20.30.50>
    DocumentRoot /www/example2
    ServerName www.example.org
    # ...
</VirtualHost>

<VirtualHost 172.20.30.40>
    DocumentRoot /www/example3
    ServerName www.example.net
    ServerAlias *.example.net
    # ...
</VirtualHost>
```

The vhost can now be accessed through the new address (as an IP-based vhost) and through the old address (as a name-based vhost).



We have a server with two name-based vhosts. In order to match the correct virtual host a client must send the correct `Host :` header. Old HTTP/1.0 clients do not send such a header and Apache has no clue what vhost the client tried to reach (and serves the request from the primary vhost). To provide as much backward compatibility as possible we create a primary vhost which returns a single page containing links with an URL prefix to the name-based virtual hosts.

Server configuration

```
NameVirtualHost 172.20.30.40

<VirtualHost 172.20.30.40>
  # primary vhost
  DocumentRoot /www/subdomain
  RewriteEngine On
  RewriteRule ^/.* /www/subdomain/index.html
  # ...
</VirtualHost>

<VirtualHost 172.20.30.40>
DocumentRoot /www/subdomain/sub1
  ServerName www.sub1.domain.tld
  ServerPath /sub1/
  RewriteEngine On
  RewriteRule ^(/sub1/.* ) /www/subdomain$1
  # ...
</VirtualHost>

<VirtualHost 172.20.30.40>
  DocumentRoot /www/subdomain/sub2
  ServerName www.sub2.domain.tld
  ServerPath /sub2/
  RewriteEngine On
  RewriteRule ^(/sub2/.* ) /www/subdomain$1
  # ...
</VirtualHost>
```

Due to the [ServerPath](#) directive a request to the URL `http://www.sub1.domain.tld/sub1/` is *always* served from the sub1-vhost.

A request to the URL `http://www.sub1.domain.tld/` is only served

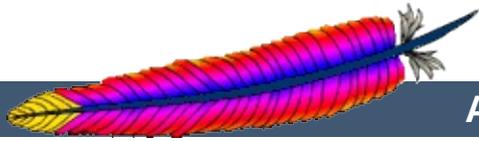
from the sub1-vhost if the client sent a correct Host : header. If no Host : header is sent the client gets the information page from the primary host.

Please note that there is one oddity: A request to `http://www.sub2.domain.tld/sub1/` is also served from the sub1-vhost if the client sent no Host : header.

The [RewriteRule](#) directives are used to make sure that a client which sent a correct Host : header can use both URL variants, *i.e.*, with or without URL prefix.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Virtual Hosts](#)

An In-Depth Discussion of Virtual Host Matching

The virtual host code was completely rewritten in **Apache 1.3**. This document attempts to explain exactly what Apache does when deciding what virtual host to serve a hit from. With the help of the new [NameVirtualHost](#) directive virtual host configuration should be a lot easier and safer than with versions prior to 1.3.

If you just want to *make it work* without understanding how, here are [some examples](#).



There is a *main_server* which consists of all the definitions appearing outside of `<VirtualHost>` sections. There are virtual servers, called *vhosts*, which are defined by `<VirtualHost>` sections.

The directives `ServerName` and `ServerPath` can appear anywhere within the definition of a server. However, each appearance overrides the previous appearance (within that server).

The *main_server* has no default `ServerPath`, or `ServerAlias`. The default `ServerName` is deduced from the server's IP address.

Port numbers specified in the `VirtualHost` directive do not influence what port numbers Apache will listen on, they only discriminate between which `VirtualHost` will be selected to handle a request.

Each address appearing in the `VirtualHost` directive can have an optional port. If the port is unspecified it is treated as a wildcard port. The special port `*` indicates a wildcard that matches any port. Collectively the entire set of addresses (including multiple A record results from DNS lookups) are called the *vhost's address set*.

Unless a `NameVirtualHost` directive is used for the exact IP address and port pair in the `VirtualHost` directive, Apache selects the best match only on the basis of the IP address (or wildcard) and port number. If there are multiple identical best matches, the first `VirtualHost` appearing in the configuration file will be selected.

If you want Apache to *further* discriminate on the basis of the HTTP Host header supplied by the client, the `NameVirtualHost` directive *must* appear with the exact IP address (or wildcard) and port pair used in a corresponding set of `VirtualHost` directives.

The name-based virtual host selection occurs only after a single IP-based virtual host has been selected, and only considers the set of virtual hosts

that carry an identical IP address and port pair.

Hostnames can be used in place of IP addresses in a virtual host definition, but it is resolved at startup and is not recommended.

Multiple `NameVirtualHost` directives can be used each with a set of `VirtualHost` directives but only one `NameVirtualHost` directive should be used for each specific IP:port pair.

The ordering of `NameVirtualHost` and `VirtualHost` directives is not important which makes the following two examples identical (only the order of the `VirtualHost` directives for *one* address set is important, see below):

```
NameVirtualHost 111.22.33.44
<VirtualHost 111.22.33.44>
# server A
...
</VirtualHost>
<VirtualHost 111.22.33.44>
# server B
...
</VirtualHost>

NameVirtualHost 111.22.33.55
<VirtualHost 111.22.33.55>
# server C
...
</VirtualHost>
<VirtualHost 111.22.33.55>
# server D
...
</VirtualHost>
```

```
<VirtualHost 111.22.33.44>
# server A
</VirtualHost>
<VirtualHost 111.22.33.55>
# server C
...
</VirtualHost>
<VirtualHost 111.22.33.44>
# server B
...
</VirtualHost>
<VirtualHost 111.22.33.55>
# server D
...
</VirtualHost>

NameVirtualHost 111.22.33.44
NameVirtualHost 111.22.33.55
```

(To aid the readability of your configuration you should prefer the left variant.)

During initialization a list for each IP address is generated and inserted

into an hash table. If the IP address is used in a `NameVirtualHost` directive the list contains all name-based vhosts for the given IP address. If there are no vhosts defined for that address the `NameVirtualHost` directive is ignored and an error is logged. For an IP-based vhost the list in the hash table is empty.

Due to a fast hashing function the overhead of hashing an IP address during a request is minimal and almost not existent. Additionally the table is optimized for IP addresses which vary in the last octet.

For every vhost various default values are set. In particular:

1. If a vhost has no `ServerAdmin`, `Timeout`, `KeepAliveTimeout`, `KeepAlive`, `MaxKeepAliveRequests`, `ReceiveBufferSize`, or `SendBufferSize` directive then the respective value is inherited from the `main_server`. (That is, inherited from whatever the final setting of that value is in the `main_server`.)
2. The "lookup defaults" that define the default directory permissions for a vhost are merged with those of the `main_server`. This includes any per-directory configuration information for any module.
3. The per-server configs for each module from the `main_server` are merged into the vhost server.

Essentially, the `main_server` is treated as "defaults" or a "base" on which to build each vhost. But the positioning of these `main_server` definitions in the config file is largely irrelevant -- the entire config of the `main_server` has been parsed when this final merging occurs. So even if a `main_server` definition appears after a vhost definition it might affect the vhost definition.

If the `main_server` has no `ServerName` at this point, then the hostname of the machine that `httpd` is running on is used instead. We will call the *main_server address set* those IP addresses returned by a DNS lookup on the `ServerName` of the `main_server`.

For any undefined `ServerName` fields, a name-based vhost defaults to the address given first in the `VirtualHost` statement defining the vhost.

Any vhost that includes the magic `_default_` wildcard is given the same `ServerName` as the `main_server`.



Virtual Host Matching

The server determines which vhost to use for a request as follows:

Hash table lookup

When the connection is first made by a client, the IP address to which the client connected is looked up in the internal IP hash table.

If the lookup fails (the IP address wasn't found) the request is served from the `_default_` vhost if there is such a vhost for the port to which the client sent the request. If there is no matching `_default_` vhost the request is served from the `main_server`.

If the IP address is not found in the hash table then the match against the port number may also result in an entry corresponding to a `NameVirtualHost *`, which is subsequently handled like other name-based vhosts.

If the lookup succeeded (a corresponding list for the IP address was found) the next step is to decide if we have to deal with an IP-based or a name-base vhost.

IP-based vhost

If the entry we found has an empty name list then we have found an IP-based vhost, no further actions are performed and the request is served from that vhost.

Name-based vhost

If the entry corresponds to a name-based vhost the name list contains one or more vhost structures. This list contains the vhosts in the same order as the `VirtualHost` directives appear in the config file.

The first vhost on this list (the first vhost in the config file with the specified IP address) has the highest priority and catches any request to

an unknown server name or a request without a Host : header field.

If the client provided a Host : header field the list is searched for a matching vhost and the first hit on a ServerName or ServerAlias is taken and the request is served from that vhost. A Host : header field can contain a port number, but Apache always matches against the real port to which the client sent the request.

The complete list of names in the VirtualHost directive are treated just like a (non wildcard) ServerAlias (but are not overridden by any ServerAlias statement).

If the client submitted a HTTP/1.0 request without Host : header field we don't know to what server the client tried to connect and any existing ServerPath is matched against the URI from the request. The first matching path on the list is used and the request is served from that vhost.

If no matching vhost could be found the request is served from the first vhost with a matching port number that is on the list for the IP to which the client connected (as already mentioned before).

Persistent connections

The IP lookup described above is only done *once* for a particular TCP/IP session while the name lookup is done on *every* request during a KeepAlive/persistent connection. In other words a client may request pages from different name-based vhosts during a single persistent connection.

Absolute URI

If the URI from the request is an absolute URI, and its hostname and port match the main server or one of the configured virtual hosts *and* match the address and port to which the client sent the request, then the

scheme/hostname/port prefix is stripped off and the remaining relative URI is served by the corresponding main server or virtual host. If it does not match, then the URI remains untouched and the request is taken to be a proxy request.

Observations

- A name-based vhost can never interfere with an IP-based vhost and vice versa. IP-based vhosts can only be reached through an IP address of its own address set and never through any other address. The same applies to name-based vhosts, they can only be reached through an IP address of the corresponding address set which must be defined with a `NameVirtualHost` directive.
- `ServerAlias` and `ServerPath` checks are never performed for an IP-based vhost.
- The order of name-/IP-based, the `_default_` vhost and the `NameVirtualHost` directive within the config file is not important. Only the ordering of name-based vhosts for a specific address set is significant. The one name-based vhost that comes first in the configuration file has the highest priority for its corresponding address set.
- The `Host :` header field is never used during the matching process. Apache always uses the real port to which the client sent the request.
- If a `ServerPath` directive exists which is a prefix of another `ServerPath` directive that appears later in the configuration file, then the former will always be matched and the latter will never be matched. (That is assuming that no `Host :` header field was available to disambiguate the two.)
- If two IP-based vhosts have an address in common, the vhost appearing first in the config file is always matched. Such a thing might happen inadvertently. The server will give a warning in the error logfile when it detects this.
- A `_default_` vhost catches a request only if there is no other vhost

with a matching IP address *and* a matching port number for the request. The request is only caught if the port number to which the client sent the request matches the port number of your `_default_` vhost which is your standard `Listen` by default. A wildcard port can be specified (*i.e.*, `_default_:*`) to catch requests to any available port. This also applies to `NameVirtualHost *` vhosts. Note that this is simply an extension of the "best match" principle, as a specific and exact match is favored over a wildcard.

- The `main_server` is only used to serve a request if the IP address and port number to which the client connected is unspecified and does not match any other vhost (including a `_default_` vhost). In other words the `main_server` only catches a request for an unspecified address/port combination (unless there is a `_default_` vhost which matches that port).
- A `_default_` vhost or the `main_server` is *never* matched for a request with an unknown or missing `Host :` header field if the client connected to an address (and port) which is used for name-based vhosts, *e.g.*, in a `NameVirtualHost` directive.
- You should never specify DNS names in `VirtualHost` directives because it will force your server to rely on DNS to boot. Furthermore it poses a security threat if you do not control the DNS for all the domains listed. There's [more information](#) available on this and the next two topics.
- `ServerName` should always be set for each vhost. Otherwise a DNS lookup is required for each vhost.



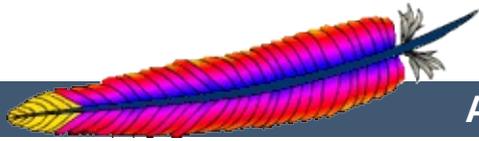
Tips

In addition to the tips on the [DNS Issues](#) page, here are some further tips:

- Place all `main_server` definitions before any `VirtualHost` definitions. (This is to aid the readability of the configuration -- the post-config merging process makes it non-obvious that definitions mixed in around virtual hosts might affect all virtual hosts.)
- Group corresponding `NameVirtualHost` and `VirtualHost` definitions in your configuration to ensure better readability.
- Avoid `ServerPaths` which are prefixes of other `ServerPaths`. If you cannot avoid this then you have to ensure that the longer (more specific) prefix vhost appears earlier in the configuration file than the shorter (less specific) prefix (*i.e.*, "`ServerPath /abc`" should appear after "`ServerPath /abc/def`").

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Virtual Hosts](#)

File Descriptor Limits

When using a large number of Virtual Hosts, Apache may run out of available file descriptors (sometimes called *file handles*) if each Virtual Host specifies different log files. The total number of file descriptors used by Apache is one for each distinct error log file, one for every other log file directive, plus 10-20 for internal use. Unix operating systems limit the number of file descriptors that may be used by a process; the limit is typically 64, and may usually be increased up to a large hard-limit.

Although Apache attempts to increase the limit as required, this may not work if:

1. Your system does not provide the `setrlimit()` system call.
2. The `setrlimit(RLIMIT_NOFILE)` call does not function on your system (such as Solaris 2.3)
3. The number of file descriptors required exceeds the hard limit.
4. Your system imposes other limits on file descriptors, such as a limit on stdio streams only using file descriptors below 256. (Solaris 2)

In the event of problems you can:

- Reduce the number of log files; don't specify log files in the `<VirtualHost>` sections, but only log to the main log files. (See [Splitting up your log files](#), below, for more information on doing this.)
- If you system falls into 1 or 2 (above), then increase the file descriptor limit before starting Apache, using a script like

```
#!/bin/sh
ulimit -S -n 100
exec httpd
```



Splitting up your log files

If you want to log multiple virtual hosts to the same log file, you may want to split up the log files afterwards in order to run statistical analysis of the various virtual hosts. This can be accomplished in the following manner.

First, you will need to add the virtual host information to the log entries. This can be done using the [LogFormat](#) directive, and the %v variable. Add this to the beginning of your log format string:

```
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost  
CustomLog logs/multiple_vhost_log vhost
```

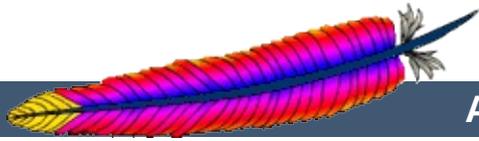
This will create a log file in the common log format, but with the canonical virtual host (whatever appears in the [ServerName](#) directive) prepended to each line. (See [mod_log_config](#) for more about customizing your log files.)

When you wish to split your log file into its component parts (one file per virtual host) you can use the program [split-logfile](#) to accomplish this. You'll find this program in the support directory of the Apache distribution.

Run this program with the command:

```
split-logfile < /logs/multiple_vhost_log
```

This program, when run with the name of your vhost log file, will generate one file for each virtual host that appears in your log file. Each file will be called `hostname.log`.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_rewrite`

Description:	Provides a rule-based rewriting engine to rewrite requested URLs on the fly
Status:	Extension
Module Identifier:	<code>rewrite_module</code>
Source File:	<code>mod_rewrite.c</code>
Compatibility:	Available in Apache 1.3 and later

Summary

This module uses a rule-based rewriting engine (based on a regular-expression parser) to rewrite requested URLs on the fly. It supports an unlimited number of rules and an unlimited number of attached rule conditions for each rule, to provide a really flexible and powerful URL manipulation mechanism. The URL manipulations can depend on various tests, of server variables, environment variables, HTTP headers, or time stamps. Even external database lookups in various formats can be used to achieve highly granular URL matching.

This module operates on the full URLs (including the path-info part) both in per-server context (`httpd.conf`) and per-directory context (`.htaccess`) and can generate query-string parts on result. The rewritten result can lead to internal sub-processing, external request redirection or even to an internal proxy throughput.

Further details, discussion, and examples, are provided in the [detailed `mod_rewrite` documentation](#).

See also

[Rewrite Flags](#)



Escaping Special Characters

As of Apache 1.3.20, special characters in *TestString* and *Substitution* strings can be escaped (that is, treated as normal characters without their usual special meaning) by prefixing them with a backslash ('\') character. In other words, you can include an actual dollar-sign character in a *Substitution* string by using '\\$'; this keeps `mod_rewrite` from trying to treat it as a backreference.



This module keeps track of two additional (non-standard) CGI/SSI environment variables named SCRIPT_URL and SCRIPT_URI. These contain the *logical* Web-view to the current resource, while the standard CGI/SSI variables SCRIPT_NAME and SCRIPT_FILENAME contain the *physical* System-view.

Notice: These variables hold the URI/URL *as they were initially requested*, that is, *before* any rewriting. This is important to note because the rewriting process is primarily used to rewrite logical URLs to physical pathnames.

Example

```
SCRIPT_NAME=/sw/lib/w3s/tree/global/u/rse/.www/index.html  
SCRIPT_FILENAME=/u/rse/.www/index.html  
SCRIPT_URL=/u/rse/  
SCRIPT_URI=http://en1.engelschall.com/u/rse/
```



Configuring Virtual Hosts

By default, `mod_rewrite` configuration settings from the main server context are not inherited by virtual hosts. To make the main server settings apply to virtual hosts, you must place the following directives in each `<VirtualHost>` section:

```
RewriteEngine On  
RewriteOptions Inherit
```



For numerous examples of common, and not-so-common, uses for `mod_rewrite`, see the [extended rewrite documentation](#).



Description:	Sets the base URL for per-directory rewrites
Syntax:	<code>RewriteBase URL -path</code>
Default:	None
Context:	directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteBase` directive specifies the URL prefix to be used for per-directory (htaccess) `RewriteRule` directives that substitute a relative path.

This directive is *required* when you use a relative path in a substitution in per-directory (htaccess) context unless either of the following conditions are true:

- The original request, and the substitution, are underneath the `DocumentRoot` (as opposed to reachable by other means, such as `Alias`).
- The *filesystem* path to the directory containing the `RewriteRule`, suffixed by the relative substitution is also valid as a URL path on the server (this is rare).

In the example below, `RewriteBase` is necessary to avoid rewriting to `http://example.com/opt/myapp-1.2.3/welcome.html` since the resource was not relative to the document root. This misconfiguration would normally cause the server to look for an "opt" directory under the document root.

```
DocumentRoot /var/www/example.com
Alias /myapp /opt/myapp-1.2.3
<Directory /opt/myapp-1.2.3>
RewriteEngine On
RewriteBase /myapp/
```

```
RewriteRule ^index\.html$ welcome.html  
</Directory>
```



Description:	Defines a condition under which rewriting will take place
Syntax:	<code>RewriteCond TestString CondPattern</code>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteCond` directive defines a rule condition. One or more `RewriteCond` can precede a `RewriteRule` directive. The following rule is then only used if both the current state of the URI matches its pattern, **and** if these conditions are met.

`TestString` is a string which can contain the following expanded constructs in addition to plain text:

- **RewriteRule backreferences:** These are backreferences of the form `$N` ($0 \leq N \leq 9$), which provide access to the grouped parts (in parentheses) of the pattern, from the `RewriteRule` which is subject to the current set of `RewriteCond` conditions..
- **RewriteCond backreferences:** These are backreferences of the form `%N` ($1 \leq N \leq 9$), which provide access to the grouped parts (again, in parentheses) of the pattern, from the last matched `RewriteCond` in the current set of conditions.
- **RewriteMap expansions:** These are expansions of the form `${mapname:key|default}`. See [the documentation for RewriteMap](#) for more details.
- **Server-Variables:** These are variables of the form `%{NAME_OF_VARIABLE}` where `NAME_OF_VARIABLE` can be a string taken from the following list:

HTTP headers:

**connection &
request:**

HTTP_USER_AGENT	REMOTE_ADDR	
HTTP_REFERER	REMOTE_HOST	
HTTP_COOKIE	REMOTE_PORT	
HTTP_FORWARDED	REMOTE_USER	
HTTP_HOST	REMOTE_IDENT	
HTTP_PROXY_CONNECTION	REQUEST_METHOD	
HTTP_ACCEPT	SCRIPT_FILENAME	
	PATH_INFO	
	QUERY_STRING	
	AUTH_TYPE	
server internals:	date and time:	specials:
DOCUMENT_ROOT	TIME_YEAR	API_VERSION
SERVER_ADMIN	TIME_MON	THE_REQUEST
SERVER_NAME	TIME_DAY	REQUEST_URI
SERVER_ADDR	TIME_HOUR	REQUEST_FILENAME
SERVER_PORT	TIME_MIN	IS_SUBREQ
SERVER_PROTOCOL	TIME_SEC	HTTPS
SERVER_SOFTWARE	TIME_WDAY	
	TIME	

These variables all correspond to the similarly named HTTP MIME-headers, C variables of the Apache server or struct tm fields of the Unix system. Most are documented elsewhere in the Manual or in the CGI specification.

SERVER_NAME and SERVER_PORT depend on the values of [UseCanonicalName](#) and [UseCanonicalPhysicalPort](#) respectively.

Those that are special to mod_rewrite include those below.

IS_SUBREQ

Will contain the text "true" if the request currently being processed is a sub-request, "false" otherwise. Sub-requests

may be generated by modules that need to resolve additional files or URIs in order to complete their tasks.

API_VERSION

This is the version of the Apache module API (the internal interface between server and module) in the current httpd build, as defined in include/ap_mmn.h. The module API version corresponds to the version of Apache in use (in the release version of Apache 1.3.14, for instance, it is 19990320:10), but is mainly of interest to module authors.

THE_REQUEST

The full HTTP request line sent by the browser to the server (e.g., "GET /index.html HTTP/1.1"). This does not include any additional headers sent by the browser. This value has not been unescaped (decoded), unlike most other variables below.

REQUEST_URI

The path component of the requested URI, such as "/index.html". This notably excludes the query string which is available as its own variable named QUERY_STRING.

REQUEST_FILENAME

The full local filesystem path to the file or script matching the request, if this has already been determined by the server at the time REQUEST_FILENAME is referenced. Otherwise, such as when used in virtual host context, the same value as REQUEST_URI.

HTTPS

Will contain the text "on" if the connection is using SSL/TLS, or "off" otherwise. (This variable can be safely used regardless of whether or not [mod_ssl](#) is loaded).

Other things you should be aware of:

1. The variables `SCRIPT_FILENAME` and `REQUEST_FILENAME` contain the same value - the value of the `filename` field of the internal `request_rec` structure of the Apache server. The first name is the commonly known CGI variable name while the second is the appropriate counterpart of `REQUEST_URI` (which contains the value of the `uri` field of `request_rec`).

If a substitution occurred and the rewriting continues, the value of both variables will be updated accordingly.

If used in per-server context (*i.e.*, before the request is mapped to the filesystem) `SCRIPT_FILENAME` and `REQUEST_FILENAME` cannot contain the full local filesystem path since the path is unknown at this stage of processing. Both variables will initially contain the value of `REQUEST_URI` in that case. In order to obtain the full local filesystem path of the request in per-server context, use an URL-based look-ahead `%{LA-U:REQUEST_FILENAME}` to determine the final value of `REQUEST_FILENAME`.

2. `%{ENV:variable}`, where *variable* can be any environment variable, is also available. This is looked-up via internal Apache structures and (if not found there) via `getenv()` from the Apache server process.
3. `%{SSL:variable}`, where *variable* is the name of an [SSL environment variable](#), can be used whether or not `mod_ssl` is loaded, but will always expand to the empty string if it is not. Example: `%{SSL:SSL_CIPHER_USEKEYSIZE}` may expand to 128. These variables are available even without setting the `StdEnvVars` option of the [SSLOptions](#) directive.
4. `%{HTTP:header}`, where *header* can be any HTTP MIME-header name, can always be used to obtain the value of a header sent in the HTTP request. Example: `%{HTTP:Proxy-Connection}` is the value of the HTTP header `Proxy-Connection`.

If a HTTP header is used in a condition this header is added to the Vary header of the response in case the condition evaluates to true for the request. It is **not** added if the condition evaluates to false for the request. Adding the HTTP header to the Vary header of the response is needed for proper caching.

It has to be kept in mind that conditions follow a short circuit logic in the case of the '**ornext |OR**' flag so that certain conditions might not be evaluated at all.

5. `%{LA-U:variable}` can be used for look-aheads which perform an internal (URL-based) sub-request to determine the final value of *variable*. This can be used to access variable for rewriting which is not available at the current stage, but will be set in a later phase. For instance, to rewrite according to the `REMOTE_USER` variable from within the per-server context (`httpd.conf` file) you must use `%{LA-U:REMOTE_USER}` - this variable is set by the authorization phases, which come *after* the URL translation phase (during which `mod_rewrite` operates).

On the other hand, because `mod_rewrite` implements its per-directory context (`.htaccess` file) via the Fixup phase of the API and because the authorization phases come *before* this phase, you just can use `%{REMOTE_USER}` in that context.

6. `%{LA-F:variable}` can be used to perform an internal (filename-based) sub-request, to determine the final value of *variable*. Most of the time, this is the same as LA-U above.

CondPattern is the condition pattern, a regular expression which is applied to the current instance of the *TestString*. *TestString* is first evaluated, before being matched against *CondPattern*.

Remember: *CondPattern* is a *perl compatible regular expression* with some additions:

1. You can prefix the pattern string with a '!' character (exclamation mark) to specify a **non**-matching pattern.
2. There are some special variants of *CondPatterns*. Instead of real regular expression strings you can also use one of the following:
 - '**<CondPattern**' (lexicographically precedes)
Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* lexicographically precedes *CondPattern*.
 - '**>CondPattern**' (lexicographically follows)
Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* lexicographically follows *CondPattern*.
 - '**=CondPattern**' (lexicographically equal)
Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* is lexicographically equal to *CondPattern* (the two strings are exactly equal, character for character). If *CondPattern* is "" (two quotation marks) this compares *TestString* to the empty string.
 - '**-d**' (is **directory**)
Treats the *TestString* as a pathname and tests whether or not it exists, and is a directory.
 - '**-f**' (is regular **file**)
Treats the *TestString* as a pathname and tests whether or not it exists, and is a regular file.
 - '**-s**' (is regular file, with **size**)
Treats the *TestString* as a pathname and tests whether or not it exists, and is a regular file with size greater than zero.
 - '**-l**' (is symbolic **link**)
Treats the *TestString* as a pathname and tests whether or not it exists, and is a symbolic link.
 - '**-x**' (has **executable** permissions)

Treats the *TestString* as a pathname and tests whether or not it exists, and has executable permissions. These permissions are determined according to the underlying OS.

- **'-F'** (is existing file, via subrequest)
Checks whether or not *TestString* is a valid file, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!
- **'-U'** (is existing URL, via subrequest)
Checks whether or not *TestString* is a valid URL, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!

This flag *only* returns information about things like access control, authentication, and authorization. This flag *does not* return information about the status code the configured handler (static file, CGI, proxy, etc.) would have returned.

Note:

All of these tests can also be prefixed by an exclamation mark (!) to negate their meaning.

3. You can also set special flags for *CondPattern* by appending **[flags]** as the third argument to the `RewriteCond` directive, where *flags* is a comma-separated list of any of the following flags:

- **'nocase|NC'** (no case)
This makes the test case-insensitive - differences between 'A-Z' and 'a-z' are ignored, both in the expanded *TestString* and the *CondPattern*. This flag is effective only for comparisons between *TestString* and *CondPattern*. It has no effect on filesystem and subrequest checks.

- **'ornext | OR'** (or next condition)

Use this to combine rule conditions with a local OR instead of the implicit AND. Typical example:

```
RewriteCond %{REMOTE_HOST} =host1 [OR]
RewriteCond %{REMOTE_HOST} =host2 [OR]
RewriteCond %{REMOTE_HOST} =host3
RewriteRule ...some special stuff for any of these hosts...
```

Without this flag you would have to write the condition/rule pair three times.

- **'novary | NV'** (no vary)

If a HTTP header is used in the condition, this flag prevents this header from being added to the Vary header of the response. Using this flag might break proper caching of the response if the representation of this response varies on the value of this header. So this flag should be only used if the meaning of the Vary header is well understood.

Example:

To rewrite the Homepage of a site according to the ``User-Agent :'' header of the request, you can use the following:

```
RewriteCond %{HTTP_USER_AGENT} ^Mozilla
RewriteRule ^/$ /homepage.max.html [L]

RewriteCond %{HTTP_USER_AGENT} ^Lynx
RewriteRule ^/$ /homepage.min.html [L]

RewriteRule ^/$ /homepage.std.html [L]
```

Explanation: If you use a browser which identifies itself as 'Mozilla' (including Netscape Navigator, Mozilla etc), then you get the max homepage (which could include frames, or other special features). If you use the Lynx browser (which is terminal-based), then you get the min

homepage (which could be a version designed for easy, text-only browsing). If neither of these conditions apply (you use any other browser, or your browser identifies itself as something non-standard), you get the std (standard) homepage.



RewriteEngine Directive

Description:	Enables or disables runtime rewriting engine
Syntax:	RewriteEngine on off
Default:	RewriteEngine off
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteEngine` directive enables or disables the runtime rewriting engine. If it is set to `off` this module does no runtime processing at all. It does not even update the `SCRIPT_URx` environment variables.

Use this directive to disable the module instead of commenting out all the `RewriteRule` directives!

Note that rewrite configurations are not inherited by virtual hosts. This means that you need to have a `RewriteEngine on` directive for each virtual host in which you wish to use rewrite rules.

`RewriteMap` directives of the type `prg` are not started during server initialization if they're defined in a context that does not have `RewriteEngine` set to `on`



Description:	Sets the name of the lock file used for RewriteMap synchronization
Syntax:	<code>RewriteLock <i>file-path</i></code>
Context:	server config
Status:	Extension
Module:	<code>mod_rewrite</code>

This directive sets the filename for a synchronization lockfile which `mod_rewrite` needs to communicate with [RewriteMap](#) programs. Set this lockfile to a local path (not on a NFS-mounted device) when you want to use a rewriting map-program. It is not required for other types of rewriting maps.



RewriteLog Directive

Description:	Sets the name of the file used for logging rewrite engine processing
Syntax:	RewriteLog <i>file-path pipe</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_rewrite

The **RewriteLog** directive sets the name of the file to which the server logs any rewriting actions it performs. If the name does not begin with a slash ('/') then it is assumed to be relative to the *Server Root*. The directive should occur only once per server config.

To disable the logging of rewriting actions it is not recommended to set *Filename* to `/dev/null`, because although the rewriting engine does not then output to a logfile it still creates the logfile output internally.

This will slow down the server with no advantage to the administrator! To disable logging either remove or comment out the **RewriteLog** directive or use `RewriteLogLevel 0!`

The **RewriteLog** log file format is as follows:

Description	Example
Remote host IP address	192.168.200.166
Remote login name	Will usually be "-"
HTTP user auth name	Username, or "-" if no auth
Date and time of request	[28/Aug/2009:13:09:09 --0400]
Virtualhost and virtualhost ID	[www.example.com/sid#84a650]
Request ID, and whether it's a subrequest	[rid#9f0e58/subreq]
Log entry severity level	(2)
Text error message	forcing proxy-throughput with

Security

See the [Apache Security Tips](#) document for details on how your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

Example

```
# Log to a file:  
RewriteLog "/usr/local/var/apache/logs/rewrite.log"  
  
# Log to a pipe:  
RewriteLog "|/path/to/parser.pl"
```



Description:	Sets the verbosity of the log file used by the rewrite engine
Syntax:	<code>RewriteLogLevel <i>Level</i></code>
Default:	<code>RewriteLogLevel 0</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_rewrite</code>

The `RewriteLogLevel` directive sets the verbosity level of the rewriting logfile. The default level 0 means no logging, while 9 or more means that practically all actions are logged.

To disable the logging of rewriting actions simply set *Level* to 0. This disables all rewrite action logs.

Using a high value for *Level* will slow down your Apache server dramatically! Use the rewriting logfile at a *Level* greater than 2 only for debugging!

Example

```
RewriteLogLevel 3
```



Description:	Defines a mapping function for key-lookup
Syntax:	<code>RewriteMap MapName MapType:MapSource</code>
Context:	server config, virtual host
Status:	Extension
Module:	mod_rewrite
Compatibility:	The choice of different dbm types is available in Apache 2.0.41 and later

The `RewriteMap` directive defines a *Rewriting Map* which can be used inside rule substitution strings by the mapping-functions to insert/substitute fields through a key lookup. The source of this lookup can be of various types.

The *MapName* is the name of the map and will be used to specify a mapping-function for the substitution strings of a rewriting rule via one of the following constructs:

```
#{ MapName : LookupKey }
#{ MapName : LookupKey | DefaultValue }
```

When such a construct occurs, the map *MapName* is consulted and the key *LookupKey* is looked-up. If the key is found, the map-function construct is substituted by *SubstValue*. If the key is not found then it is substituted by *DefaultValue* or by the empty string if no *DefaultValue* was specified. Empty values behave as if the key was absent, therefore it is not possible to distinguish between empty-valued keys and absent keys.

For example, you might define a `RewriteMap` as:

```
RewriteMap examplemap txt:/path/to/file/map.txt
```

You would then be able to use this map in a `RewriteRule` as follows:

```
RewriteRule ^/ex/(.*) ${examplemap:$1}
```

The following combinations for *MapType* and *MapSource* can be used:

- **Standard Plain Text**

MapType: txt, MapSource: Unix filesystem path to valid regular file
This is the standard rewriting map feature where the *MapSource* is a plain ASCII file containing either blank lines, comment lines (starting with a '#' character) or pairs like the following - one per line.

MatchingKey SubstValue

Example

```
##  
## map.txt -- rewriting map  
##  
  
Ralf.S.Engelschall    rse    # Bastard Operator From Hell  
Mr.Joe.Average       joe    # Mr. Average
```

```
RewriteMap real-to-user txt:/path/to/file/map.txt
```

- **Randomized Plain Text**

MapType: rnd, MapSource: Unix filesystem path to valid regular file
This is identical to the Standard Plain Text variant above but with a special post-processing feature: After looking up a value it is parsed according to contained ``|'' characters which have the meaning of ``or''. In other words they indicate a set of alternatives from which the actual returned value is chosen randomly. For example, you might use the following map file and directives to provide a random load balancing between several back-end servers, via a reverse-proxy. Images are sent to one of the servers in the 'static' pool, while everything else is sent to one of the 'dynamic' pool.

Example:

Rewrite map file

```
##
##  map.txt -- rewriting map
##

static  www1|www2|www3|www4
dynamic www5|www6
```

Configuration directives

```
RewriteMap servers rnd:/path/to/file/map.txt

RewriteRule ^/(.*\.(png|gif|jpg)) http://${servers:static}/$1
[NC,P,L]
RewriteRule ^/(.*) http://${servers:dynamic}/$1 [P,L]
```

- **Hash File**

MapType: `dbm [=type]`, MapSource: Unix filesystem path to valid regular file

Here the source is a binary format DBM file containing the same contents as a *Plain Text* format file, but in a special representation which is optimized for really fast lookups. The *type* can be `sdbm`, `gdbm`, `ndbm`, or `db` depending on [compile-time settings](#). If the *type* is omitted, the compile-time default will be chosen.

To create a dbm file from a source text file, use the [httxt2dbm](#) utility.

```
$ htxt2dbm -i mapfile.txt -o mapfile.map
```

- **Internal Function**

MapType: `int`, MapSource: Internal Apache function

Here, the source is an internal Apache function. Module authors can provide additional internal functions by registering them with the

ap_register_rewrite_mapfunc API. The functions that are provided by default are:

- **toupper:**
Converts the key to all upper case.
- **tolower:**
Converts the key to all lower case.
- **escape:**
Translates special characters in the key to hex-encodings.
- **unescape:**
Translates hex-encodings in the key back to special characters.

- **External Rewriting Program**

MapType: prg, MapSource: Unix filesystem path to valid regular file
Here the source is a program, not a map file. To create it you can use a language of your choice, but the result has to be an executable program (either object-code or a script with the magic cookie trick '#!/path/to/interpreter' as the first line).

This program is started once, when the Apache server is started, and then communicates with the rewriting engine via its `stdin` and `stdout` file-handles. For each map-function lookup it will receive the key to lookup as a newline-terminated string on `stdin`. It then has to give back the looked-up value as a newline-terminated string on `stdout` or the four-character string ```NULL"` if it fails (*i.e.*, there is no corresponding value for the given key). A trivial program which will implement a 1:1 map (*i.e.*, `key == value`) could be:

External rewriting programs are not started if they're defined in a context that does not have `RewriteEngine` set to on.

```
#!/usr/bin/perl
$| = 1;
while (<STDIN>) {
```

```
# ...put here any transformations or lookups...
print $_;
}
```

But be very careful:

1. *"Keep it simple, stupid"* (KISS). If this program hangs, it will cause Apache to hang when trying to use the relevant rewrite rule.
2. A common mistake is to use buffered I/O on `stdout`. Avoid this, as it will cause a deadlock! ```$|=1"` is used above, to prevent this.
3. The `RewriteLock` directive can be used to define a lockfile which `mod_rewrite` can use to synchronize communication with the mapping program. By default no such synchronization takes place.

The `RewriteMap` directive can occur more than once. For each mapping-function use one `RewriteMap` directive to declare its rewriting mapfile. While you cannot **declare** a map in per-directory context it is of course possible to **use** this map in per-directory context.

Note

For plain text and DBM format files the looked-up keys are cached in-core until the `mtime` of the mapfile changes or the server does a restart. This way you can have map-functions in rules which are used for **every** request. This is no problem, because the external lookup only happens once!



Description:	Sets some special options for the rewrite engine
Syntax:	RewriteOptions <i>Options</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite
Compatibility:	MaxRedirects is no longer available in version 2.1 and later

The `RewriteOptions` directive sets some special options for the current per-server or per-directory configuration. The *Option* string can currently only be one of the following:

inherit

This forces the current configuration to inherit the configuration of the parent. In per-virtual-server context, this means that the maps, conditions and rules of the main server are inherited. In per-directory context this means that conditions and rules of the parent directory's .htaccess configuration are inherited.

Rules inherited from the parent scope are applied **after** rules specified in the child scope.

AllowAnyURI

When `RewriteRule` is used in `VirtualHost` or server context with version 2.2.23 or later of httpd, `mod_rewrite` will only process the rewrite rules if the request URI is a [URL-path](#). This avoids some security issues where particular rules could allow "surprising" pattern expansions (see [CVE-2011-3368](#) and [CVE-2011-4317](#)). To lift the restriction on matching a URL-path, the `AllowAnyURI` option can be enabled, and `mod_rewrite` will apply the rule set to any request URI string, regardless of whether that string matches the URL-path

grammar required by the HTTP specification.

Security Warning

Enabling this option will make the server vulnerable to security issues if used with rewrite rules which are not carefully authored. It is **strongly recommended** that this option is not used. In particular, beware of input strings containing the '@' character which could change the interpretation of the transformed URI, as per the above CVE names.

MergeBase

With this option, the value of [RewriteBase](#) is copied from where it's explicitly defined into any sub-directory or sub-location that doesn't define its own [RewriteBase](#). Not copying was the default until 2.2.22. In version 2.2.23 copying was the default. The flag to explicitly control it is available for Apache HTTP Server 2.2.24 and later.



Description:	Defines rules for the rewriting engine
Syntax:	RewriteRule <i>Pattern Substitution</i> [<i>flags</i>]
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The **RewriteRule** directive is the real rewriting workhorse. The directive can occur more than once, with each instance defining a single rewrite rule. The order in which these rules are defined is important - this is the order in which they will be applied at run-time.

Pattern is a perl compatible regular expression. On the first RewriteRule it is applied to the (%-decoded) [URL-path](#) of the request; subsequent patterns are applied to the output of the last matched RewriteRule.

What is matched?

In **VirtualHost** context, The *Pattern* will initially be matched against the part of the URL after the hostname and port, and before the query string (e.g. "/app1/index.html").

In **Directory** and htaccess context, the *Pattern* will initially be matched against the *filesystem* path, after removing the prefix that led the server to the current **RewriteRule** (e.g. "app1/index.html" or "index.html" depending on where the directives are defined).

If you wish to match against the hostname, port, or query string, use a **RewriteCond** with the `#{HTTP_HOST}`, `#{SERVER_PORT}`, or `#{QUERY_STRING}` variables respectively.

Per-directory Rewrites

- The rewrite engine may be used in [.htaccess](#) files and in

[<Directory>](#) sections, with some additional complexity.

- To enable the rewrite engine in this context, you need to set "RewriteEngine On" **and** "Options FollowSymLinks" must be enabled. If your administrator has disabled override of FollowSymLinks for a user's directory, then you cannot use the rewrite engine. This restriction is required for security reasons.
- When using the rewrite engine in .htaccess files the per-directory prefix (which always is the same for a specific directory) is automatically *removed* for the RewriteRule pattern matching and automatically *added* after any relative (not starting with a slash or protocol name) substitution encounters the end of a rule set. See the [RewriteBase](#) directive for more information regarding what prefix will be added back to relative substitutions.
- If you wish to match against the full URL-path in a per-directory (htaccess) RewriteRule, use the %{REQUEST_URI} variable in a [RewriteCond](#).
- The removed prefix always ends with a slash, meaning the matching occurs against a string which *never* has a leading slash. Therefore, a *Pattern* with ^/ never matches in per-directory context.
- Although rewrite rules are syntactically permitted in [<Location>](#) and [<Files>](#) sections, this should never be necessary and is unsupported.

For some hints on [regular expressions](#), see the [mod_rewrite Introduction](#).

In mod_rewrite, the NOT character ('!') is also available as a possible pattern prefix. This enables you to negate a pattern; to say, for instance: *`if the current URL does **NOT** match this pattern"*. This can be used for exceptional cases, where it is easier to match the negative pattern, or as a last default rule.

Note

When using the NOT character to negate a pattern, you cannot include grouped wildcard parts in that pattern. This is because, when the pattern does NOT match (ie, the negation matches), there are no contents for the groups. Thus, if negated patterns are used, you cannot use \$N in the substitution string!

The *Substitution* of a rewrite rule is the string that replaces the original URL-path that was matched by *Pattern*. The *Substitution* may be a:

file-system path

Designates the location on the file-system of the resource to be delivered to the client. Substitutions are only treated as a file-system path when the rule is configured in server (virtualhost) context and the first component of the path in the substitution exists in the file-system

URL-path

A [DocumentRoot](#)-relative path to the resource to be served. Note that [mod_rewrite](#) tries to guess whether you have specified a file-system path or a URL-path by checking to see if the first segment of the path exists at the root of the file-system. For example, if you specify a *Substitution* string of `/www/file.html`, then this will be treated as a URL-path *unless* a directory named `www` exists at the root of your file-system (or, in the case of using rewrites in a `.htaccess` file, relative to your document root), in which case it will be treated as a file-system path. If you wish other URL-mapping directives (such as [Alias](#)) to be applied to the resulting URL-path, use the `[PT]` flag as described below.

Absolute URL

If an absolute URL is specified, [mod_rewrite](#) checks to see whether the hostname matches the current host. If it does, the scheme and hostname are stripped out and the resulting path is treated as a URL-path. Otherwise, an external redirect is performed for the given URL. To force an external redirect back to the current

host, see the [R] flag below.

- (dash)

A dash indicates that no substitution should be performed (the existing path is passed through untouched). This is used when a flag (see below) needs to be applied without changing the path.

In addition to plain text, the *Substitution* string can include

1. back-references ($\$N$) to the RewriteRule pattern
2. back-references ($\%N$) to the last matched RewriteCond pattern
3. server-variables as in rule condition test-strings ($\%\{\text{VARNAME}\}$)
4. [mapping-function](#) calls ($\$\{\text{mapname}:\text{key}|\text{default}\}$)

Back-references are identifiers of the form $\$N$ ($N=0..9$), which will be replaced by the contents of the N th group of the matched *Pattern*. The server-variables are the same as for the *TestString* of a RewriteCond directive. The mapping-functions come from the RewriteMap directive and are explained there. These three types of variables are expanded in the order above.

Rewrite rules are applied to the results of previous rewrite rules, in the order in which they are defined in the config file. The URL is **completely replaced** by the *Substitution* and the rewriting process continues until all rules have been applied, or it is explicitly terminated by a [L flag](#), or other flag which implies immediate termination, such as **F**.

Modifying the Query String

By default, the query string is passed through unchanged. You can, however, create URLs in the substitution string containing a query string part. Simply use a question mark inside the substitution string to indicate that the following text should be re-injected into the query string. When you want to erase an existing query string, end the

substitution string with just a question mark. To combine new and old query strings, use the [QSA] flag.

Additionally you can set special actions to be performed by appending **[flags]** as the third argument to the RewriteRule directive. *Flags* is a comma-separated list, surround by square brackets, of any of the flags in the following table. More details, and examples, for each flag, are available in the [Rewrite Flags document](#).

Flag and syntax	Function
B	Escape non-alphanumeric characters <i>before</i> app transformation. details ...
chain C	Rule is chained to the following rule. If the rule fa rule(s) chained to it will be skipped. details ...
cookie CO=NAME:VAL	Sets a cookie in the client browser. Full syntax is: CO=NAME:VAL:domain[:lifetime[:path[:secure[:h details ...
discardpath DPI	Causes the PATH_INFO portion of the rewritten U discarded. details ...
env E=[!]VAR[:VAL]	Causes an environment variable VAR to be set (t value VAL if provided). The form !VAR causes the environment variable VAR to be unset. details ...
forbidden F	Returns a 403 FORBIDDEN response to the clier browser. details ...
gone G	Returns a 410 GONE response to the client brow details ...
Handler H=Content-handler	Causes the resulting URI to be sent to the specifi Content-handler for processing. details ...
last L	Stop the rewriting process immediately and don't any more rules. Especially note caveats for per-d and .htaccess context. details ...
next N	Re-run the rewriting process, starting again with t

	rule, using the result of the ruleset so far as a starting point. details ...
nocase NC	Makes the pattern comparison case-insensitive. details ...
noescape NE	Prevent mod_rewrite from applying hexcode escapes to special characters in the result of the rewrite. details ...
nosubreq NS	Causes a rule to be skipped if the current request is an internal sub-request. details ...
proxy P	Force the substitution URL to be internally sent as a proxy request. details ...
passthrough PT	Forces the resulting URI to be passed back to the mapping engine for processing of other URI-to-file translators, such as Alias or Redirect. details ...
qsappend QSA	Appends any query string from the original request to any query string created in the rewrite target. details ...
redirect R[= <i>code</i>]	Forces an external redirect, optionally with the specified HTTP status code. details ...
skip S= <i>num</i>	Tells the rewriting engine to skip the next <i>num</i> rules if the current rule matches. details ...
type T= <i>MIME-type</i>	Force the MIME-type of the target file to be the specified type. details ...

Home directory expansion

When the substitution string begins with a string resembling "/~user" (via explicit text or backreferences), mod_rewrite performs home directory expansion independent of the presence or configuration of [mod_userdir](#).

This expansion does not occur when the *PT* flag is used on the [RewriteRule](#) directive.

Here are all possible substitution combinations and their meanings:

**Inside per-server configuration (httpd.conf)
for request ``GET /somepath/pathinfo``:**

Given Rule	Resulting Substitution
^/somepath(.*) otherpath\$1	invalid, not supported
^/somepath(.*) otherpath\$1 [R]	invalid, not supported
^/somepath(.*) otherpath\$1 [P]	invalid, not supported
^/somepath(.*) /otherpath\$1	/otherpath/pathinfo
^/somepath(.*) /otherpath\$1 [R]	http://thishost/otherpa via external redirectio
^/somepath(.*) /otherpath\$1 [P]	doesn't make sense, not
^/somepath(.*) http://thishost/otherpath\$1	/otherpath/pathinfo
^/somepath(.*) http://thishost/otherpath\$1 [R]	http://thishost/otherpa via external redirectio
^/somepath(.*) http://thishost/otherpath\$1 [P]	doesn't make sense, not
^/somepath(.*) http://otherhost/otherpath\$1	http://otherhost/otherp via external redirectio
^/somepath(.*) http://otherhost/otherpath\$1 [R]	http://otherhost/otherp via external redirectio (the [R] flag is redund
^/somepath(.*) http://otherhost/otherpath\$1 [P]	http://otherhost/otherp via internal proxy

**Inside per-directory configuration for /somepath
(/physical/path/to/somepath/.htaccess, with RewriteBase
/somepath)
for request ``GET /somepath/localpath/pathinfo``:**

Given Rule	Resulting Substitution
-----	-----

<code>^localpath(.*) otherpath\$1</code>	<code>/somepath/otherpath/pat</code>
<code>^localpath(.*) otherpath\$1 [R]</code>	<code>http://thishost/somepat</code> via external redirectio
<code>^localpath(.*) otherpath\$1 [P]</code>	doesn't make sense, not

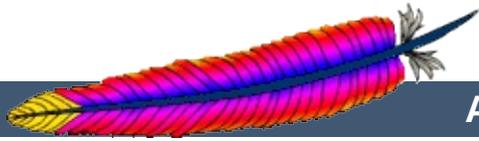
<code>^localpath(.*) /otherpath\$1</code>	<code>/otherpath/pathinfo</code>
<code>^localpath(.*) /otherpath\$1 [R]</code>	<code>http://thishost/otherpa</code> via external redirectio
<code>^localpath(.*) /otherpath\$1 [P]</code>	doesn't make sense, not

<code>^localpath(.*) http://thishost/otherpath\$1</code>	<code>/otherpath/pathinfo</code>
<code>^localpath(.*) http://thishost/otherpath\$1 [R]</code>	<code>http://thishost/otherpa</code> via external redirectio
<code>^localpath(.*) http://thishost/otherpath\$1 [P]</code>	doesn't make sense, not

<code>^localpath(.*) http://otherhost/otherpath\$1</code>	<code>http://otherhost/otherp</code> via external redirectio
<code>^localpath(.*) http://otherhost/otherpath\$1 [R]</code>	<code>http://otherhost/otherp</code> via external redirectio (the [R] flag is redund
<code>^localpath(.*) http://otherhost/otherpath\$1 [P]</code>	<code>http://otherhost/otherp</code> via internal proxy

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Apache mod_rewrite Introduction

This document supplements the [mod_rewrite reference documentation](#). It describes the basic concepts necessary for use of [mod_rewrite](#). Other documents go into greater detail, but this doc should help the beginner get their feet wet.

See also

[Module documentation](#)

[Redirection and remapping](#)

[Controlling access](#)

[Virtual hosts](#)

[Proxying](#)

[Using RewriteMap](#)

[Advanced techniques](#)

[When not to use mod_rewrite](#)



The Apache module [mod_rewrite](#) is a very powerful and sophisticated module which provides a way to do URL manipulations. With it, you can do nearly all types of URL rewriting that you may need. It is, however, somewhat complex, and may be intimidating to the beginner. There is also a tendency to treat rewrite rules as magic incantation, using them without actually understanding what they do.

This document attempts to give sufficient background so that what follows is understood, rather than just copied blindly.

Remember that many common URL-manipulation tasks don't require the full power and complexity of [mod_rewrite](#). For simple tasks, see [mod_alias](#) and the documentation on [mapping URLs to the filesystem](#).

Finally, before proceeding, be sure to configure [mod_rewrite](#)'s log level to one of the trace levels using the [LogLevel](#) directive. Although this can give an overwhelming amount of information, it is indispensable in debugging problems with [mod_rewrite](#) configuration, since it will tell you exactly how each rule is processed.



mod_rewrite uses the [Perl Compatible Regular Expression](#) vocabulary. In this document, we do not attempt to provide a detailed reference to regular expressions. For that, we recommend the [PCRE man pages](#), the [Perl regular expression man page](#), and [Mastering Regular Expressions, by Jeffrey Friedl](#).

In this document, we attempt to provide enough of a regex vocabulary to get you started, without being overwhelming, in the hope that [RewriteRules](#) will be scientific formulae, rather than magical incantations.

Regex vocabulary

The following are the minimal building blocks you will need, in order to write regular expressions and [RewriteRules](#). They certainly do not represent a complete regular expression vocabulary, but they are a good place to start, and should help you read basic regular expressions, as well as write your own.

Character	Meaning	Example
.	Matches any single character	c.t will match cat, cot, cut, etc.
+	Repeats the previous match one or more times	a+ matches a, aa, aaa, etc
*	Repeats the previous match zero or more times.	a* matches all the same things a+ matches, but will also match an empty string.
?	Makes the match optional.	colou?r will match color and colour.
^	Called an anchor, matches the beginning of the string	^a matches a string that begins with a
\$	The other anchor, this	a\$ matches a string that ends

	matches the end of the string.	with a.
()	Groups several characters into a single unit, and captures a match for use in a backreference.	(ab)+ matches ababab - that is, the + applies to the group. For more on backreferences see below .
[]	A character class - matches one of the characters	c[uoa]t matches cut, cot or cat.
[^]	Negative character class - matches any character not specified	c[^/]t matches cat or c=t but not c/t

In [mod_rewrite](#) the ! character can be used before a regular expression to negate it. This is, a string will be considered to have matched only if it does not match the rest of the expression.

Regex Back-Reference Availability

One important thing here has to be remembered: Whenever you use parentheses in *Pattern* or in one of the *CondPattern*, back-references are internally created which can be used with the strings \$N and %N (see below). These are available for creating the strings *Substitution* and *TestString* as outlined in the following chapters. Figure 1 shows to which locations the back-references are transferred for expansion as well as illustrating the flow of the RewriteRule, RewriteCond matching. In the next chapters, we will be exploring how to use these back-references, so do not fret if it seems a bit alien to you at first.

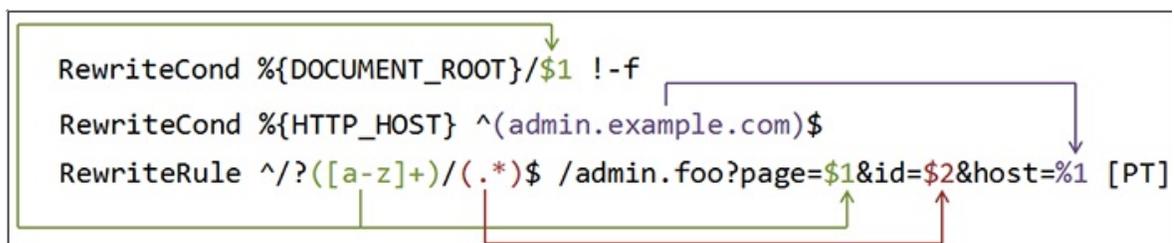


Figure 1: The back-reference flow through a rule.

*In this example, a request for /test/1234 would be transformed into /admin.foo?
page=test&id=1234&host=admin.example.com.*



A **RewriteRule** consists of three arguments separated by spaces. The arguments are

1. *Pattern*: which incoming URLs should be affected by the rule;
2. *Substitution*: where should the matching requests be sent;
3. *[flags]*: options affecting the rewritten request.

The *Pattern* is a **regular expression**. It is initially (for the first rewrite rule or until a substitution occurs) matched against the URL-path of the incoming request (the part after the hostname but before any question mark indicating the beginning of a query string) or, in per-directory context, against the request's path relative to the directory for which the rule is defined. Once a substitution has occurred, the rules that follow are matched against the substituted value.

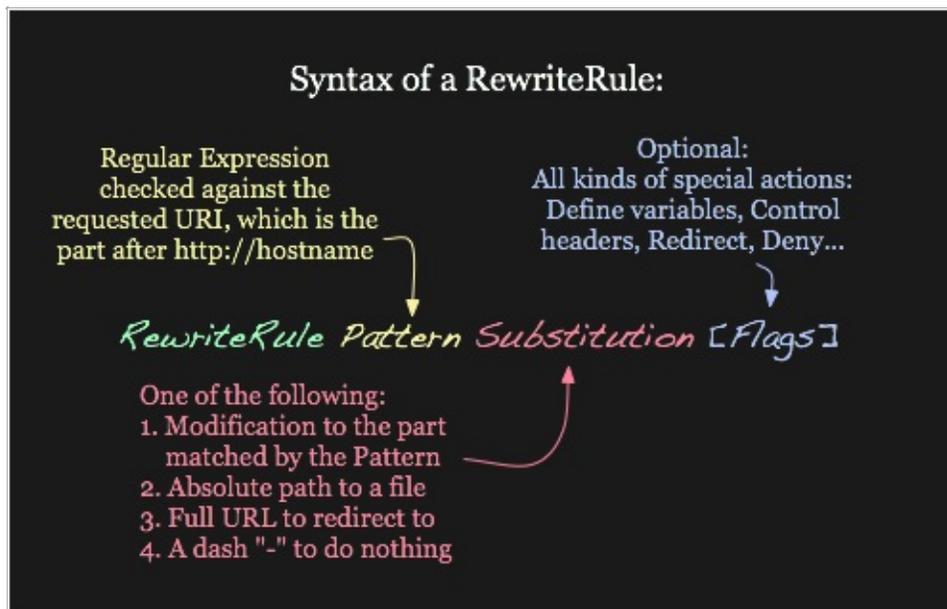


Figure 2: Syntax of the RewriteRule directive.

The *Substitution* can itself be one of three things:

A full filesystem path to a resource

```
RewriteRule ^/games.* /usr/local/games/web
```

This maps a request to an arbitrary location on your filesystem, much like the [Alias](#) directive.

A web-path to a resource

```
RewriteRule ^/foo$ /bar
```

If [DocumentRoot](#) is set to `/usr/local/apache2/htdocs`, then this directive would map requests for `http://example.com/foo` to the path `/usr/local/apache2/htdocs/bar`.

An absolute URL

```
RewriteRule ^/product/view$  
http://site2.example.com/seeproduct.html [R]
```

This tells the client to make a new request for the specified URL.

The *Substitution* can also contain *back-references* to parts of the incoming URL-path matched by the *Pattern*. Consider the following:

```
RewriteRule ^/product/(.*)/view$ /var/web/productdb/$1
```

The variable `$1` will be replaced with whatever text was matched by the expression inside the parenthesis in the *Pattern*. For example, a request for `http://example.com/product/r14df/view` will be mapped to the path `/var/web/productdb/r14df`.

If there is more than one expression in parenthesis, they are available in order in the variables `$1`, `$2`, `$3`, and so on.



Rewrite Flags

The behavior of a [RewriteRule](#) can be modified by the application of one or more flags to the end of the rule. For example, the matching behavior of a rule can be made case-insensitive by the application of the [NC] flag:

```
RewriteRule ^puppy.html smalldog.html [NC]
```

For more details on the available flags, their meanings, and examples, see the [Rewrite Flags](#) document.



One or more [RewriteCond](#) directives can be used to restrict the types of requests that will be subject to the following [RewriteRule](#). The first argument is a variable describing a characteristic of the request, the second argument is a [regular expression](#) that must match the variable, and a third optional argument is a list of flags that modify how the match is evaluated.

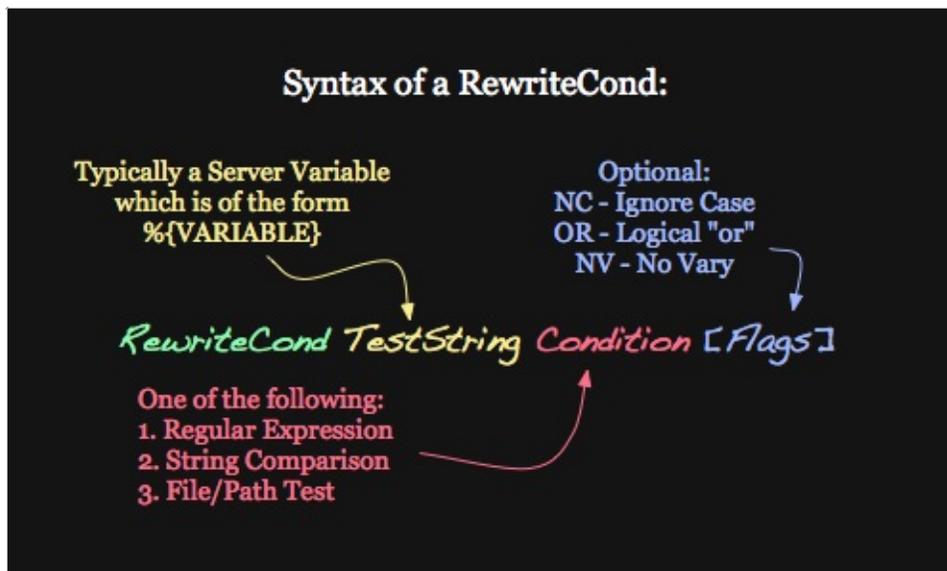


Figure 3: Syntax of the RewriteCond directive

For example, to send all requests from a particular IP range to a different server, you could use:

```
RewriteCond %{REMOTE_ADDR} ^10\.2\  
RewriteRule (.*) http://intranet.example.com$1
```

When more than one [RewriteCond](#) is specified, they must all match for the [RewriteRule](#) to be applied. For example, to deny requests that contain the word "hack" in their query string, unless they also contain a cookie containing the word "go", you could use:

```
RewriteCond %{QUERY_STRING} hack  
RewriteCond %{HTTP_COOKIE} !go  
RewriteRule .* - [F]
```

Notice that the exclamation mark specifies a negative match, so the rule is only applied if the cookie does not contain "go".

Matches in the regular expressions contained in the [RewriteConds](#) can be used as part of the *Substitution* in the [RewriteRule](#) using the variables %1, %2, etc. For example, this will direct the request to a different directory depending on the hostname used to access the site:

```
RewriteCond %{HTTP_HOST} (!.*)  
RewriteRule ^/(.*) /sites/%1/$1
```

If the request was for `http://example.com/foo/bar`, then %1 would contain `example.com` and \$1 would contain `foo/bar`.



The `RewriteMap` directive provides a way to call an external function, so to speak, to do your rewriting for you. This is discussed in greater detail in the [RewriteMap supplementary documentation](#).

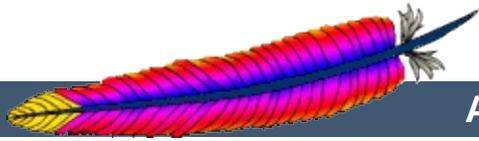


Rewriting is typically configured in the main server configuration setting (outside any [<Directory>](#) section) or inside [<VirtualHost>](#) containers. This is the easiest way to do rewriting and is recommended. It is possible, however, to do rewriting inside [<Directory>](#) sections or [.htaccess files](#) at the expense of some additional complexity. This technique is called per-directory rewrites.

The main difference with per-server rewrites is that the path prefix of the directory containing the .htaccess file is stripped before matching in the [RewriteRule](#). In addition, the [RewriteBase](#) should be used to assure the request is properly mapped.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Redirecting and Remapping with `mod_rewrite`

This document supplements the [mod_rewrite reference documentation](#). It describes how you can use `mod_rewrite` to redirect and remap request. This includes many examples of common uses of `mod_rewrite`, including detailed descriptions of how each works.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

- [Module documentation](#)
- [mod_rewrite introduction](#)
- [Controlling access](#)
- [Virtual hosts](#)
- [Proxying](#)
- [Using RewriteMap](#)
- [Advanced techniques and tricks](#)
- [When not to use mod_rewrite](#)



Description:

Assume we have recently renamed the page `foo.html` to `bar.html` and now want to provide the old URL for backward compatibility. However, we want that users of the old URL even not recognize that the pages was renamed - that is, we don't want the address to change in their browser.

Solution:

We rewrite the old URL to the new one internally via the following rule:

```
RewriteEngine on
RewriteRule ^/foo\.html$ /bar.html [PT]
```



Description:

Assume again that we have recently renamed the page `foo.html` to `bar.html` and now want to provide the old URL for backward compatibility. But this time we want that the users of the old URL get hinted to the new one, i.e. their browsers Location field should change, too.

Solution:

We force a HTTP redirect to the new URL which leads to a change of the browsers and thus the users view:

```
RewriteEngine on
RewriteRule ^/foo\.html$ bar.html [R]
```

Discussion

In this example, as contrasted to the [internal](#) example above, we can simply use the Redirect directive. `mod_rewrite` was used in that earlier example in order to hide the redirect from the client:

```
Redirect /foo.html /bar.html
```



Description:

If a resource has moved to another server, you may wish to have URLs continue to work for a time on the old server while people update their bookmarks.

Solution:

You can use [mod_rewrite](#) to redirect these URLs to the new server, but you might also consider using the `Redirect` or `RedirectMatch` directive.

With `mod_rewrite`

```
RewriteEngine on  
RewriteRule ^/docs/(.+) http://new.example.com/docs/$1 [R,L]
```

With `RedirectMatch`

```
RedirectMatch ^/docs/(.*) http://new.example.com/docs/$1
```

With `Redirect`

```
Redirect /docs/ http://new.example.com/docs/
```



Description:

How can we transform a static page `foo.html` into a dynamic variant `foo.cgi` in a seamless way, i.e. without notice by the browser/user.

Solution:

We just rewrite the URL to the CGI-script and force the handler to be **cgi-script** so that it is executed as a CGI program. This way a request to `/~quux/foo.html` internally leads to the invocation of `/~quux/foo.cgi`.

```
RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo\.html$ foo.cgi [H=cgi-script]
```



Description:

How can we make URLs backward compatible (still existing virtually) after migrating document .YYYY to document .XXXX, e.g. after translating a bunch of .html files to .php?

Solution:

We rewrite the name to its basename and test for existence of the new extension. If it exists, we take that name, else we rewrite the URL to its original state.

```
# backward compatibility ruleset for
# rewriting document.html to document.php
# when and only when document.php exists
<Directory /var/www/htdocs>
  RewriteEngine on
  RewriteBase /var/www/htdocs

  RewriteCond $1.php -f
  RewriteCond $1.html !-f
  RewriteRule ^(.*)\.html$ $1.php
</Directory>
```

Discussion

This example uses an often-overlooked feature of `mod_rewrite`, by taking advantage of the order of execution of the ruleset. In particular, `mod_rewrite` evaluates the left-hand-side of the `RewriteRule` before it evaluates the `RewriteCond` directives. Consequently, `$1` is already defined by the time the `RewriteCond` directives are evaluated. This allows us to test for the existence of the original (document .html) and target (document .php) files using the same base filename.

This ruleset is designed to use in a per-directory context (In a `<Directory>` block or in a .htaccess file), so that the `-f` checks are looking at the correct directory path. You may need to set a [RewriteBase](#) directive to specify the directory base that you're

working in.



Description:

The goal of this rule is to force the use of a particular hostname, in preference to other hostnames which may be used to reach the same site. For example, if you wish to force the use of **www.example.com** instead of **example.com**, you might use a variant of the following recipe.

Solution:

The very best way to solve this doesn't involve `mod_rewrite` at all, but rather uses the `Redirect` directive placed in a virtual host for the non-canonical hostname(s).

```
<VirtualHost *:80>
  ServerName undesired.example.com
  ServerAlias example.com notthis.example.com

  Redirect / http://www.example.com/
</VirtualHost>

<VirtualHost *:80>
  ServerName www.example.com
</VirtualHost>
```

If, for whatever reason, you still want to use `mod_rewrite` - if, for example, you need this to work with a larger set of `RewriteRules` - you might use one of the recipes below.

For sites running on a port other than 80:

```
RewriteCond %{HTTP_HOST} !^www\.example\.com [NC]
RewriteCond %{HTTP_HOST} !^$
RewriteCond %{SERVER_PORT} !^80$
RewriteRule ^/?(.*) http://www.example.com:%{SERVER_PORT}/$1
[L,R,NE]
```

And for a site running on port 80

```
RewriteCond %{HTTP_HOST} !^www\.example\.com [NC]
```

```
RewriteCond %{HTTP_HOST} !^$  
RewriteRule ^/?(.*) http://www.example.com/$1 [L,R,NE]
```

If you wanted to do this generically for all domain names - that is, if you want to redirect **example.com** to **www.example.com** for all possible values of **example.com**, you could use the following recipe:

```
RewriteCond %{HTTP_HOST} !^www\.. [NC]  
RewriteCond %{HTTP_HOST} !^$  
RewriteRule ^/?(.*) http://www.%{HTTP_HOST}/$1 [L,R,NE]
```

These rulesets will work either in your main server configuration file, or in a `.htaccess` file placed in the [DocumentRoot](#) of the server.



Description:

A particular resource might exist in one of several places, and we want to look in those places for the resource when it is requested. Perhaps we've recently rearranged our directory structure, dividing content into several locations.

Solution:

The following ruleset searches in two directories to find the resource, and, if not finding it in either place, will attempt to just serve it out of the location requested.

```
RewriteEngine on

# first try to find it in dir1/...
# ...and if found stop and be happy:
RewriteCond %{DOCUMENT_ROOT}/dir1/%{REQUEST_URI} -f
RewriteRule ^(.+) %{DOCUMENT_ROOT}/dir1/$1 [L]

# second try to find it in dir2/...
# ...and if found stop and be happy:
RewriteCond %{DOCUMENT_ROOT}/dir2/%{REQUEST_URI} -f
RewriteRule ^(.+) %{DOCUMENT_ROOT}/dir2/$1 [L]

# else go on for other Alias or ScriptAlias directives,
# etc.
RewriteRule ^ - [PT]
```



Description:

We have numerous mirrors of our website, and want to redirect people to the one that is located in the country where they are located.

Solution:

Looking at the hostname of the requesting client, we determine which country they are coming from. If we can't do a lookup on their IP address, we fall back to a default server.

We'll use a [RewriteMap](#) directive to build a list of servers that we wish to use.

```
HostnameLookups on
RewriteEngine on
RewriteMap multiplex txt:/path/to/map.mirrors
RewriteCond %{REMOTE_HOST} ([a-z]+)$ [NC]
RewriteRule ^/(.*)$ ${multiplex:%1|http://www.example.com/}$1
[R,L]
```

```
## map.mirrors -- Multiplexing Map

de http://www.example.de/
uk http://www.example.uk/
com http://www.example.com/
##EOF##
```

Discussion

This ruleset relies on [HostNameLookups](#) being set on, which can be a significant performance hit.

The [RewriteCond](#) directive captures the last portion of the hostname of the requesting client - the country code - and the following RewriteRule uses that value to look up the appropriate mirror host in the map file.



Description:

We wish to provide different content based on the browser, or user-agent, which is requesting the content.

Solution:

We have to decide, based on the HTTP header "User-Agent", which content to serve. The following config does the following: If the HTTP header "User-Agent" contains "Mozilla/3", the page `foo.html` is rewritten to `foo.NS.html` and the rewriting stops. If the browser is "Lynx" or "Mozilla" of version 1 or 2, the URL becomes `foo.20.html`. All other browsers receive page `foo.32.html`. This is done with the following ruleset:

```
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/3.*
RewriteRule ^foo\.html$ foo.NS.html [L]

RewriteCond %{HTTP_USER_AGENT} ^Lynx/ [OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/[12]
RewriteRule ^foo\.html$ foo.20.html [L]

RewriteRule ^foo\.html$ foo.32.html [L]
```



Description:

On some web servers there is more than one URL for a resource. Usually there are canonical URLs (which are actually used and distributed) and those which are just shortcuts, internal ones, and so on. Independent of which URL the user supplied with the request, they should finally see the canonical one in their browser address bar.

Solution:

We do an external HTTP redirect for all non-canonical URLs to fix them in the location view of the Browser and for all subsequent requests. In the example ruleset below we replace `/puppies` and `/canines` by the canonical `/dogs`.

```
RewriteRule ^/(puppies|canines)/(.*) /dogs/$2 [R]
```

Discussion:

This should really be accomplished with `Redirect` or `RedirectMatch` directives:

```
RedirectMatch ^/(puppies|canines)/(.*) /dogs/$2
```



Description:

Usually the [DocumentRoot](#) of the webserver directly relates to the URL "/". But often this data is not really of top-level priority. For example, you may wish for visitors, on first entering a site, to go to a particular subdirectory /about/. This may be accomplished using the following ruleset:

Solution:

We redirect the URL / to /about/:

```
RewriteEngine on
RewriteRule ^/$ /about/ [R]
```

Note that this can also be handled using the [RedirectMatch](#) directive:

```
RedirectMatch ^/$ http://example.com/about/
```

Note also that the example rewrites only the root URL. That is, it rewrites a request for `http://example.com/`, but not a request for `http://example.com/page.html`. If you have in fact changed your document root - that is, if **all** of your content is in fact in that subdirectory, it is greatly preferable to simply change your [DocumentRoot](#) directive, or move all of the content up one directory, rather than rewriting URLs.



Description:

You want a single resource (say, a certain file, like `index.php`) to handle all requests that come to a particular directory, except those that should go to an existing resource such as an image, or a css file.

Solution:

As of version 2.2.16, you should use the [FallbackResource](#) directive for this:

```
<Directory /var/www/my_blog>
  FallbackResource index.php
</Directory>
```

However, in earlier versions of Apache, or if your needs are more complicated than this, you can use a variation of the following rewrite set to accomplish the same thing:

```
<Directory /var/www/my_blog>
  RewriteBase /my_blog

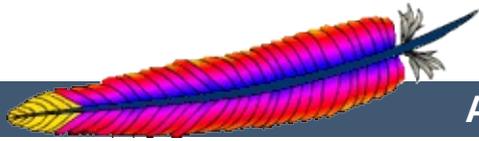
  RewriteCond /var/www/my_blog/%{REQUEST_FILENAME} !-f
  RewriteCond /var/www/my_blog/%{REQUEST_FILENAME} !-d
  RewriteRule ^ index.php [PT]
</Directory>
```

If, on the other hand, you wish to pass the requested URI as a query string argument to `index.php`, you can replace that `RewriteRule` with:

```
RewriteRule (.*) index.php?$1 [PT,QSA]
```

Note that these rulesets can be used in a `.htaccess` file, as well as in a `<Directory>` block.

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Using mod_rewrite to control access

This document supplements the [mod_rewrite reference documentation](#). It describes how you can use [mod_rewrite](#) to control access to various resources, and other related techniques. This includes many examples of common uses of mod_rewrite, including detailed descriptions of how each works.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

- [Module documentation](#)
- [mod_rewrite introduction](#)
- [Redirection and remapping](#)
- [Virtual hosts](#)
- [Proxying](#)
- [Using RewriteMap](#)
- [Advanced techniques and tricks](#)
- [When not to use mod_rewrite](#)



Description:

The following technique forbids the practice of other sites including your images inline in their pages. This practice is often referred to as "hotlinking", and results in your bandwidth being used to serve content for someone else's site.

Solution:

This technique relies on the value of the HTTP_REFERER variable, which is optional. As such, it's possible for some people to circumvent this limitation. However, most users will experience the failed request, which should, over time, result in the image being removed from that other site.

There are several ways that you can handle this situation.

In this first example, we simply deny the request, if it didn't initiate from a page on our site. For the purpose of this example, we assume that our site is `www.example.com`.

```
RewriteCond %{HTTP_REFERER} !^$  
RewriteCond %{HTTP_REFERER} !www.example.com [NC]  
RewriteRule \.(gif|jpg|png)$ - [F,NC]
```

In this second example, instead of failing the request, we display an alternate image instead.

```
RewriteCond %{HTTP_REFERER} !^$  
RewriteCond %{HTTP_REFERER} !www.example.com [NC]  
RewriteRule \.(gif|jpg|png)$ /images/go-away.png [R,NC]
```

In the third example, we redirect the request to an image on some other site.

```
RewriteCond %{HTTP_REFERER} !^$  
RewriteCond %{HTTP_REFERER} !www.example.com [NC]
```

```
RewriteRule \.(gif|jpg|png)$ http://other.example.com/image.gif  
[R,NC]
```

Of these techniques, the last two tend to be the most effective in getting people to stop hotlinking your images, because they will simply not see the image that they expected to see.

Discussion:

If all you wish to do is deny access to the resource, rather than redirecting that request elsewhere, this can be accomplished without the use of `mod_rewrite`:

```
SetEnvIf Referer example\.com localreferer  
<FilesMatch \.(jpg|png|gif)$>  
Order deny,allow  
Deny from all  
Allow from env=localreferer  
</FilesMatch>
```



Description:

In this recipe, we discuss how to block persistent requests from a particular robot, or user agent.

The standard for robot exclusion defines a file, `/robots.txt` that specifies those portions of your website where you which to exclude robots. However, some robots do not honor these files.

Note that there are methods of accomplishing this which do not use `mod_rewrite`. Note also that any technique that relies on the clients `USER_AGENT` string can be circumvented very easily, since that string can be changed.

Solution:

We use a ruleset that specifies the directory to be protected, and the client `USER_AGENT` that identifies the malicious or persistent robot.

In this example, we are blocking a robot called `NameOfBadRobot` from a location `/secret/files`. You may also specify an IP address range, if you are trying to block that user agent only from the particular source.

```
RewriteCond %{HTTP_USER_AGENT} ^NameOfBadRobot
RewriteCond %{REMOTE_ADDR} =123\.45\.67\.[8-9]
RewriteRule ^/secret/files/ - [F]
```

Discussion:

Rather than using `mod_rewrite` for this, you can accomplish the same end using alternate means, as illustrated here:

```
SetEnvIfNoCase User-Agent ^NameOfBadRobot goaway
<Location /secret/files>
Order allow,deny
Allow from all
Deny from env=goaway
</Location>
```

As noted above, this technique is trivial to circumvent, by simply modifying the USER_AGENT request header. If you are experiencing a sustained attack, you should consider blocking it at a higher level, such as at your firewall.



Description:

We wish to maintain a blacklist of hosts, rather like `hosts.deny`, and have those hosts blocked from accessing our server.

Solution:

```
RewriteEngine on
RewriteMap hosts-deny txt:/path/to/hosts.deny
RewriteCond ${hosts-deny:%{REMOTE_ADDR}|NOT-FOUND} !=NOT-FOUND
[OR]
RewriteCond ${hosts-deny:%{REMOTE_HOST}|NOT-FOUND} !=NOT-FOUND
RewriteRule ^ - [F]
```

```
##
## hosts.deny
##
## ATTENTION! This is a map, not a list, even when we treat it as
## such.
## mod_rewrite parses it for key/value pairs, so at least a
## dummy value "-" must be present for each entry.
##

193.102.180.41 -
bsdti1.sdm.de -
192.76.162.40 -
```

Discussion:

The second `RewriteCond` assumes that you have `HostNameLookups` turned on, so that client IP addresses will be resolved. If that's not the case, you should drop the second `RewriteCond`, and drop the `[OR]` flag from the first `RewriteCond`.



Description:

Redirect requests based on the Referer from which the request came, with different targets per Referer.

Solution:

The following ruleset uses a map file to associate each Referer with a redirection target.

```
RewriteMap deflector txt:/path/to/deflector.map

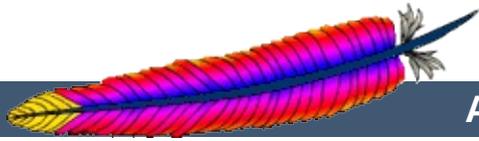
RewriteCond %{HTTP_REFERER} !=""
RewriteCond ${deflector:%{HTTP_REFERER}} =-
RewriteRule ^ %{HTTP_REFERER} [R,L]

RewriteCond %{HTTP_REFERER} !=""
RewriteCond ${deflector:%{HTTP_REFERER}|NOT-FOUND} !=NOT-FOUND
RewriteRule ^.* ${deflector:%{HTTP_REFERER}} [R,L]
```

The map file lists redirection targets for each referer, or, if we just wish to redirect back to where they came from, a "-" is placed in the map:

```
##
## deflector.map
##

http://badguys.example.com/bad/index.html -
http://badguys.example.com/bad/index2.html -
http://badguys.example.com/bad/index3.html
http://somewhere.example.com/
```



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Dynamic mass virtual hosts with `mod_rewrite`

This document supplements the [mod_rewrite reference documentation](#). It describes how you can use `mod_rewrite` to create dynamically configured virtual hosts.

`mod_rewrite` is not the best way to configure virtual hosts. You should first consider the [alternatives](#) before resorting to `mod_rewrite`. See also the "[how to avoid mod_rewrite](#)" document.

See also

- [Module documentation](#)
- [mod_rewrite introduction](#)
- [Redirection and remapping](#)
- [Controlling access](#)
- [Proxying](#)
- [RewriteMap](#)
- [Advanced techniques and tricks](#)
- [When not to use mod_rewrite](#)



Description:

We want to automatically create a virtual host for every hostname which resolves in our domain, without having to create new VirtualHost sections.

In this recipe, we assume that we'll be using the hostname `www.SITE.example.com` for each user, and serve their content out of `/home/SITE/www`.

Solution:

```
RewriteEngine on

RewriteMap lowercase int:tolower

RewriteCond %{lowercase:%{HTTP_HOST}} ^www\.[
([^.]+)\.example\.com$
RewriteRule ^(.*) /home/%1/www$1
```

Discussion

You will need to take care of the DNS resolution - Apache does not handle name resolution. You'll need either to create CNAME records for each hostname, or a DNS wildcard record. Creating DNS records is beyond the scope of this document.

The internal `tolower` RewriteMap directive is used to ensure that the hostnames being used are all lowercase, so that there is no ambiguity in the directory structure which must be created.

Parentheses used in a [RewriteCond](#) are captured into the backreferences `%1`, `%2`, etc, while parentheses used in [RewriteRule](#) are captured into the backreferences `$1`, `$2`, etc.

As with many techniques discussed in this document, `mod_rewrite`

really isn't the best way to accomplish this task. You should, instead, consider using [mod_vhost_alias](#) instead, as it will much more gracefully handle anything beyond serving static files, such as any dynamic content, and Alias resolution.



This extract from `httpd.conf` does the same thing as [the first example](#). The first half is very similar to the corresponding part above, except for some changes, required for backward compatibility and to make the `mod_rewrite` part work properly; the second half configures `mod_rewrite` to do the actual work.

Because `mod_rewrite` runs before other URI translation modules (e.g., `mod_alias`), `mod_rewrite` must be told to explicitly ignore any URLs that would have been handled by those modules. And, because these rules would otherwise bypass any `ScriptAlias` directives, we must have `mod_rewrite` explicitly enact those mappings.

```
# get the server name from the Host: header
UseCanonicalName Off

# splittable logs
LogFormat "%{Host}i %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

<Directory /www/hosts>
    # ExecCGI is needed here because we can't force
    # CGI execution in the way that ScriptAlias does
    Options FollowSymLinks ExecCGI
</Directory>

RewriteEngine On

# a ServerName derived from a Host: header may be any case at all
RewriteMap lowercase int:tolower

## deal with normal documents first:
# allow Alias /icons/ to work - repeat for other aliases
RewriteCond %{REQUEST_URI} !^/icons/
# allow CGIs to work
RewriteCond %{REQUEST_URI} !^/cgi-bin/
# do the magic
RewriteRule ^/(.*)$ /www/hosts/${lowercase:%{SERVER_NAME}}/docs/$1

## and now deal with CGIs - we have to force a handler
RewriteCond %{REQUEST_URI} ^/cgi-bin/
RewriteRule ^/(.*)$ /www/hosts/${lowercase:%{SERVER_NAME}}/cgi-bin/$1
[H=cgi-script]
```



This arrangement uses more advanced `mod_rewrite` features to work out the translation from virtual host to document root, from a separate configuration file. This provides more flexibility, but requires more complicated configuration.

The `vhost.map` file should look something like this:

```
customer-1.example.com /www/customers/1
customer-2.example.com /www/customers/2
# ...
customer-N.example.com /www/customers/N
```

The `httpd.conf` should contain the following:

```
RewriteEngine on

RewriteMap lowercase int:tolower

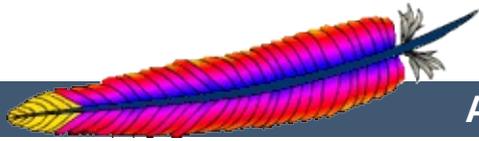
# define the map file
RewriteMap vhost txt:/www/conf/vhost.map

# deal with aliases as above
RewriteCond %{REQUEST_URI} !^/icons/
RewriteCond %{REQUEST_URI} !^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
# this does the file-based remap
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/docs/$1

RewriteCond %{REQUEST_URI} ^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/cgi-bin/$1 [H=cgi-script]
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Using mod_rewrite for Proxying

This document supplements the [mod_rewrite reference documentation](#). It describes how to use the RewriteRule's [P] flag to proxy content to another server. A number of recipes are provided that describe common scenarios.

See also

[Module documentation](#)

[mod_rewrite introduction](#)

[Redirection and remapping](#)

[Controlling access](#)

[Virtual hosts](#)

[Using RewriteMap](#)

[Advanced techniques and tricks](#)

[When not to use mod_rewrite](#)



Description:

`mod_rewrite` provides the `[P]` flag, which allows URLs to be passed, via `mod_proxy`, to another server. Two examples are given here. In one example, a URL is passed directly to another server, and served as though it were a local URL. In the other example, we proxy missing content to a back-end server.

Solution:

To simply map a URL to another server, we use the `[P]` flag, as follows:

```
RewriteEngine on
RewriteBase /products/
RewriteRule ^widget/(.*)$ http://product.example.com/widget/$1
[P]
ProxyPassReverse /products/widget/
http://product.example.com/widget/
```

In the second example, we proxy the request only if we can't find the resource locally. This can be very useful when you're migrating from one server to another, and you're not sure if all the content has been migrated yet.

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^/(.*) http://old.example.com/$1 [P]
ProxyPassReverse / http://old.example.com/
```

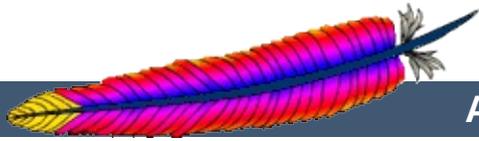
Discussion:

In each case, we add a `ProxyPassReverse` directive to ensure that any redirects issued by the backend are correctly passed on to the client.

Consider using either `ProxyPass` or `ProxyPassMatch` whenever possible in preference to `mod_rewrite`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Using RewriteMap

This document supplements the [mod_rewrite reference documentation](#). It describes the use of the [RewriteMap](#) directive, and provides examples of each of the various RewriteMap types.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

- [Module documentation](#)
- [mod_rewrite introduction](#)
- [Redirection and remapping](#)
- [Controlling access](#)
- [Virtual hosts](#)
- [Proxying](#)
- [Advanced techniques and tricks](#)
- [When not to use mod_rewrite](#)



The `RewriteMap` directive defines an external function which can be called in the context of `RewriteRule` or `RewriteCond` directives to perform rewriting that is too complicated, or too specialized to be performed just by regular expressions. The source of this lookup can be any of the types listed in the sections below, and enumerated in the `RewriteMap` reference documentation.

The syntax of the `RewriteMap` directive is as follows:

```
RewriteMap MapName MapType:MapSource
```

The *MapName* is an arbitrary name that you assign to the map, and which you will use in directives later on. Arguments are passed to the map via the following syntax:

```
${ MapName : LookupKey } ${ MapName : LookupKey |  
DefaultValue }
```

When such a construct occurs, the map *MapName* is consulted and the key *LookupKey* is looked-up. If the key is found, the map-function construct is substituted by *SubstValue*. If the key is not found then it is substituted by *DefaultValue* or by the empty string if no *DefaultValue* was specified.

For example, you might define a `RewriteMap` as:

```
RewriteMap examplemap txt:/path/to/file/map.txt
```

You would then be able to use this map in a `RewriteRule` as follows:

```
RewriteRule ^/ex/(.*) ${examplemap:$1}
```

A default value can be specified in the event that nothing is found in the map:

```
RewriteRule ^/ex/(.*) ${examplemap:$1|/not_found.html}
```

Per-directory and .htaccess context

The `RewriteMap` directive may not be used in `<Directory>` sections or `.htaccess` files. You must declare the map in server or virtualhost context. You may use the map, once created, in your `RewriteRule` and `RewriteCond` directives in those scopes. You just can't **declare** it in those scopes.

The sections that follow describe the various *MapTypes* that may be used, and give examples of each.



PLAIN-TEXT MAPS

When a MapType of txt is used, the MapSource is a filesystem path to a plain-text mapping file, containing space-separated key/value pair per line. Optionally, a line may contain a comment, starting with a '#' character.

For example, the following might be valid entries in a map file.

```
# Comment line
MatchingKey SubstValue
MatchingKey SubstValue # comment
```

When the RewriteMap is invoked the argument is looked for in the first argument of a line, and, if found, the substitution value is returned.

For example, we might use a mapfile to translate product names to product IDs for easier-to-remember URLs, using the following recipe:

Product to ID configuration

```
RewriteMap product2id txt:/etc/apache2/productmap.txt
RewriteRule ^/product/(.*) /prods.php?id=${product2id:$1|NOTFOUND}
[PT]
```

We assume here that the prods.php script knows what to do when it received an argument of id=NOTFOUND when a product is not found in the lookup map.

The file /etc/apache2/productmap.txt then contains the following:

Product to ID map

```
##
## productmap.txt - Product to ID map file
##

television 993
stereo 198
fishingrod 043
```

basketball 418
telephone 328

Thus, when `http://example.com/product/television` is requested, the `RewriteRule` is applied, and the request is internally mapped to `/prods.php?id=993`.

Note: .htaccess files

The example given is crafted to be used in server or virtualhost scope. If you're planning to use this in a `.htaccess` file, you'll need to remove the leading slash from the rewrite pattern in order for it to match anything:

```
RewriteRule ^product/(.*) /prods.php?  
id=${product2id:$1|NOTFOUND} [PT]
```

Cached lookups

The looked-up keys are cached by `httpd` until the `mtime` (modified time) of the mapfile changes, or the `httpd` server is restarted. This ensures better performance on maps that are called by many requests.



When a `MapType` of `rnd` is used, the `MapSource` is a filesystem path to a plain-text mapping file, each line of which contains a key, and one or more values separated by `|`. One of these values will be chosen at random if the key is matched.

For example, you might use the following map file and directives to provide a random load balancing between several back-end server, via a reverse-proxy. Images are sent to one of the servers in the 'static' pool, while everything else is sent to one of the 'dynamic' pool.

Rewrite map file

```
##
## map.txt -- rewriting map
##

static www1|www2|www3|www4
dynamic www5|www6
```

Configuration directives

```
RewriteMap servers rnd:/path/to/file/map.txt

RewriteRule ^/(.*\.(png|gif|jpg)) http://${servers:static}/$1 [NC,P,L]
RewriteRule ^/(.*) http://${servers:dynamic}/$1 [P,L]
```

So, when an image is requested and the first of these rules is matched, `RewriteMap` looks up the string `static` in the map file, which returns one of the specified hostnames at random, which is then used in the `RewriteRule` target.

If you wanted to have one of the servers more likely to be chosen (for example, if one of the server has more memory than the others, and so can handle more requests) simply list it more times in the map file.

```
static www1|www1|www2|www3|www4
```



When a MapType of dbm is used, the MapSource is a filesystem path to a DBM database file containing key/value pairs to be used in the mapping. This works exactly the same way as the txt map, but is much faster, because a DBM is indexed, whereas a text file is not. This allows more rapid access to the desired key.

You may optionally specify a particular dbm type:

```
RewriteMap examplemap dbm=sdbm:/etc/apache/mapfile.dbm
```

The type can be sdbm, gdbm, ndbm or db. However, it is recommended that you just use the [httxt2dbm](#) utility that is provided with Apache HTTP Server, as it will use the correct DBM library, matching the one that was used when httpd itself was built.

To create a dbm file, first create a text map file as described in the [txt](#) section. Then run `httxt2dbm`:

```
$ htxt2dbm -i mapfile.txt -o mapfile.map
```

You can then reference the resulting file in your `RewriteMap` directive:

```
RewriteMap mapname dbm:/etc/apache/mapfile.map
```

Note that with some dbm types, more than one file is generated, with a common base name. For example, you may have two files named `mapfile.map.dir` and `mapfile.map.pag`. This is normal, and you need only use the base name `mapfile.map` in your `RewriteMap` directive.

Cached lookups

The looked-up keys are cached by httpd until the `mtime` (modified

time) of the mapfile changes, or the httpd server is restarted. This ensures better performance on maps that are called by many requests.



INTERNAL FUNCTIONS

When a `MapType` of `int` is used, the `MapSource` is one of the available internal `RewriteMap` functions. Module authors can provide additional internal functions by registering them with the `ap_register_rewrite_mapfunc` API. The functions that are provided by default are:

- **toupper:**
Converts the key to all upper case.
- **tolower:**
Converts the key to all lower case.
- **escape:**
Translates special characters in the key to hex-encodings.
- **unescape:**
Translates hex-encodings in the key back to special characters.

To use one of these functions, create a `RewriteMap` referencing the `int` function, and then use that in your `RewriteRule`:

Redirect a URI to an all-lowercase version of itself

```
RewriteMap lc int:tolower
RewriteRule (.*[A-Z]+.*) ${lc:$1} [R]
```

Please note that the example offered here is for illustration purposes only, and is not a recommendation. If you want to make URLs case-insensitive, consider using [mod_spelling](#) instead.



prg: External Rewriting Program

When a MapType of prg is used, the MapSource is a filesystem path to an executable program which will providing the mapping behavior. This can be a compiled binary file, or a program in an interpreted language such as Perl or Python.

This program is started once, when the Apache HTTP Server is started, and then communicates with the rewriting engine via STDIN and STDOUT. That is, for each map function lookup, it expects one argument via STDIN, and should return one new-line terminated response string on STDOUT. If there is no corresponding lookup value, the map program should return the four-character string "NULL" to indicate this.

External rewriting programs are not started if they're defined in a context that does not have [RewriteEngine](#) set to on.

A simple example is shown here which will replace all dashes with underscores in a request URI.

Rewrite configuration

```
RewriteMap d2u prg:/www/bin/dash2under.pl
RewriteRule - ${d2u:%{REQUEST_URI}}
```

dash2under.pl

```
#!/usr/bin/perl
$| = 1; # Turn off I/O buffering
while (<STDIN>) {
    s/-/_/g; # Replace dashes with underscores
    print $_;
}
```

Use a RewriteLock!

When using a prg: RewriteMap, you should use a [RewriteLock](#). Failure to do so will result in an error message in the log file, and may result in a race condition on concurrent requests.

Caution!

- Keep your rewrite map program as simple as possible. If the program hangs, it will cause httpd to wait indefinitely for a response from the map, which will, in turn, cause httpd to stop responding to requests.
- Be sure to turn off buffering in your program. In Perl this is done by the second line in the example script: `$| = 1`; This will of course vary in other languages. Buffered I/O will cause httpd to wait for the output, and so it will hang.
- Remember that there is only one copy of the program, started at server startup. All requests will need to go through this one bottleneck. This can cause significant slowdowns if many requests must go through this process, or if the script itself is very slow.



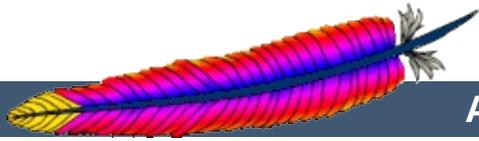
Summary

The `RewriteMap` directive can occur more than once. For each mapping-function use one `RewriteMap` directive to declare its rewriting mapfile.

While you cannot **declare** a map in per-directory context (`.htaccess` files or `<Directory>` blocks) it is possible to **use** this map in per-directory context.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Advanced Techniques with mod_rewrite

This document supplements the [mod_rewrite reference documentation](#). It provides a few advanced techniques and tricks using mod_rewrite.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

[Module documentation](#)

[mod_rewrite introduction](#)

[Redirection and remapping](#)

[Controlling access](#)

[Virtual hosts](#)

[Proxying](#)

[Using RewriteMap](#)

[When not to use mod_rewrite](#)



Description:

A common technique for distributing the burden of server load or storage space is called "sharding". When using this method, a front-end server will use the url to consistently "shard" users or objects to separate backend servers.

Solution:

A mapping is maintained, from users to target servers, in external map files. They look like:

```
user1 physical_host_of_user1
user2 physical_host_of_user2
: :
```

We put this into a map .users-to-hosts file. The aim is to map;

```
/u/user1/any path
```

to

```
http://physical_host_of_user1/u/user/any path
```

thus every URL path need not be valid on every backend physical host. The following ruleset does this for us with the help of the map files assuming that server0 is a default server which will be used if a user has no entry in the map:

```
RewriteEngine on
RewriteMap users-to-hosts txt:/path/to/map.users-to-hosts
RewriteRule ^/u/([^/]+)/?(.*) http://${users-to-hosts:$1|server0}/u/$1/$2
```

See the [RewriteMap](#) documentation for more discussion of the syntax

of this directive.



Description:

We wish to dynamically generate content, but store it statically once it is generated. This rule will check for the existence of the static file, and if it's not there, generate it. The static files can be removed periodically, if desired (say, via cron) and will be regenerated on demand.

Solution:

This is done via the following ruleset:

```
# This example is valid in per-directory cont
RewriteCond %{REQUEST_URI}    !-U
RewriteRule ^(.+)\.html$      /regenerate
```

The `-U` operator determines whether the test string (in this case, `REQUEST_URI`) is a valid URL. It does this via a subrequest. In the event that this subrequest fails - that is, the requested resource doesn't exist - this rule invokes the CGI program `/regenerate_page.cgi`, which generates the requested resource and saves it into the document directory, so that the next time it is requested, a static copy can be served.

In this way, documents that are infrequently updated can be served in static form. If documents need to be refreshed, they can be deleted from the document directory, and they will then be regenerated the next time they are requested.



Description:

We wish to randomly distribute load across several servers using `mod_rewrite`.

Solution:

We'll use [RewriteMap](#) and a list of servers to accomplish this.

```
RewriteEngine on
RewriteMap lb rnd:/path/to/serverlist.txt

RewriteRule ^/(.*) http://${lb:servers}/$1 [P,L]
```

`serverlist.txt` will contain a list of the servers:

```
## serverlist.txt

servers one.example.com|two.example.com|three.example.com
```

If you want one particular server to get more of the load than the others, add it more times to the list.

Discussion

Apache comes with a load-balancing module - [mod_proxy_balancer](#) - which is far more flexible and featureful than anything you can cobble together using `mod_rewrite`.



Description:

Wouldn't it be nice, while creating a complex web page, if the web browser would automatically refresh the page every time we save a new version from within our editor? Impossible?

Solution:

No! We just combine the MIME multipart feature, the web server NPH feature, and the URL manipulation power of [mod_rewrite](#). First, we establish a new URL feature: Adding just `:refresh` to any URL causes the 'page' to be refreshed every time it is updated on the filesystem.

```
RewriteRule ^(/[uqe]/[^\s]+/?.*):refresh /internal/cgi/apache/nph-refresh?f=$1
```

Now when we reference the URL

```
/u/foo/bar/page.html:refresh
```

this leads to the internal invocation of the URL

```
/internal/cgi/apache/nph-refresh?f=/u/foo/bar/page.html
```

The only missing part is the NPH-CGI script. Although one would usually say "left as an exercise to the reader" ;-) I will provide this, too.

```
#!/sw/bin/perl
##
##  nph-refresh -- NPH/CGI script for auto refreshing pages
##  Copyright (c) 1997 Ralf S. Engelschall, All Rights Reserved.
##
$| = 1;
```

```

# split the QUERY_STRING variable
@pairs = split( /\&/, $ENV{'QUERY_STRING'} );
foreach $pair (@pairs) {
    ( $name, $value ) = split( /=/, $pair );
    $name =~ tr/A-Z/a-z/;
    $name = 'QS_' . $name;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
    eval "\$$name = \"\$value\"";
}
$QS_s = 1 if ( $QS_s eq '' );
$QS_n = 3600 if ( $QS_n eq '' );
if ( $QS_f eq '' ) {
    print "HTTP/1.0 200 OK\n";
    print "Content-type: text/html\n\n";
print "&lt;b&gt;ERROR&lt;/b&gt;: No file given\n";
    exit(0);
}
if ( !-f $QS_f ) {
    print "HTTP/1.0 200 OK\n";
    print "Content-type: text/html\n\n";
print "&lt;b&gt;ERROR&lt;/b&gt;: File $QS_f not found\n";
    exit(0);
}

sub print_http_headers_multipart_begin {
    print "HTTP/1.0 200 OK\n";
    $bound = "ThisRandomString12345";
    print "Content-type: multipart/x-mixed-replace;boundary=$bound\n";
    &print_http_headers_multipart_next;
}

sub print_http_headers_multipart_next {
    print "\n--$bound\n";
}

```

```

}

sub print_http_headers_multipart_end {
    print "\n--$bound--\n";
}

sub displayhtml {
    local ($buffer) = @_;
    $len = length($buffer);
    print "Content-type: text/html\n";
    print "Content-length: $len\n\n";
    print $buffer;
}

sub readfile {
    local ($file) = @_;
    local ( *FP, $size, $buffer, $bytes );
    ( $x, $x, $x, $x, $x, $x, $x, $size ) = stat($file);
    $size = sprintf( "%d", $size );
    open(FP, "&lt;$file");
    $bytes = sysread( FP, $buffer, $size );
    close(FP);
    return $buffer;
}

$buffer = &readfile($QS_f);
&print_http_headers_multipart_begin;
&displayhtml($buffer);

sub mystat {
    local ($file) = $_[0];
    local ($time);

    ( $x, $mtime ) = stat($file);

```

```
        return $mtime;
    }

    $mtimeL = &mystat($QS_f);
    $mtime = $mtime;
    for ( $n = 0 ; $n < lt ; $QS_n ; $n++ ) {
        while (1) {
            $mtime = &mystat($QS_f);
            if ( $mtime ne $mtimeL ) {
                $mtimeL = $mtime;
                sleep(2);
                $buffer = &readfile($QS_f);
                &print_http_headers_multipart_next;
                &displayhtml($buffer);
                sleep(5);
                $mtimeL = &mystat($QS_f);
                last;
            }
            sleep($QS_s);
        }
    }

    &print_http_headers_multipart_end;

    exit(0);

    ##EOF##
```



Description:

Some sites with thousands of users use a structured homedir layout, *i.e.* each homedir is in a subdirectory which begins (for instance) with the first character of the username. So, `/~larry/anypath` is `/home/l/larry/public_html/anypath` while `/~waldo/anypath` is `/home/w/waldo/public_html/anypath`.

Solution:

We use the following ruleset to expand the tilde URLs into the above layout.

```
RewriteEngine on
RewriteRule ^/~(([a-z])[a-z0-9]+)(.*) /home/$2/$1/public_html$3
```



Description:

By default, redirecting to an HTML anchor doesn't work, because `mod_rewrite` escapes the `#` character, turning it into `%23`. This, in turn, breaks the redirection.

Solution:

Use the `[NE]` flag on the `RewriteRule`. `NE` stands for No Escape.

Discussion:

This technique will of course also work with other special characters that `mod_rewrite`, by default, URL-encodes.



Description:

We wish to use `mod_rewrite` to serve different content based on the time of day.

Solution:

There are a lot of variables named `TIME_XXX` for rewrite conditions. In conjunction with the special lexicographic comparison patterns `<STRING`, `>STRING` and `=STRING` we can do time-dependent redirects:

```
RewriteEngine on
RewriteCond %{TIME_HOUR}%{TIME_MIN} >0700
RewriteCond %{TIME_HOUR}%{TIME_MIN} <1900
RewriteRule ^foo\.html$ foo.day.html [L]
RewriteRule ^foo\.html$ foo.night.html
```

This provides the content of `foo.day.html` under the URL `foo.html` from `07:01-18:59` and at the remaining time the contents of `foo.night.html`.

`mod_cache`, intermediate proxies and browsers may each cache responses and cause the either page to be shown outside of the time-window configured. `mod_expires` may be used to control this effect. You are, of course, much better off simply serving the content dynamically, and customizing it based on the time of day.



Description:

At time, we want to maintain some kind of status when we perform a rewrite. For example, you want to make a note that you've done that rewrite, so that you can check later to see if a request can via that rewrite. One way to do this is by setting an environment variable.

Solution:

Use the [E] flag to set an environment variable.

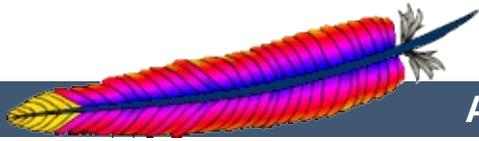
```
RewriteEngine on  
RewriteRule ^/horse/(.*) /pony/$1 [E=rewritten:1]
```

Later in your ruleset you might check for this environment variable using a RewriteCond:

```
RewriteCond %{ENV:rewritten} =1
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

When not to use mod_rewrite

This document supplements the [mod_rewrite reference documentation](#). It describes perhaps one of the most important concepts about mod_rewrite - namely, when to avoid using it.

mod_rewrite should be considered a last resort, when other alternatives are found wanting. Using it when there are simpler alternatives leads to configurations which are confusing, fragile, and hard to maintain. Understanding what other alternatives are available is a very important step towards mod_rewrite mastery.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

The most common situation in which [mod_rewrite](#) is the right tool is when the very best solution requires access to the server configuration files, and you don't have that access. Some configuration directives are only available in the server configuration file. So if you are in a hosting situation where you only have .htaccess files to work with, you may need to resort to [mod_rewrite](#).

See also

- [Module documentation](#)
- [mod_rewrite introduction](#)
- [Redirection and remapping](#)
- [Controlling access](#)
- [Virtual hosts](#)
- [Proxying](#)
- [Using RewriteMap](#)

[Advanced techniques and tricks](#)



`mod_alias` provides the `Redirect` and `RedirectMatch` directives, which provide a means to redirect one URL to another. This kind of simple redirection of one URL, or a class of URLs, to somewhere else, should be accomplished using these directives rather than `RewriteRule`. `RedirectMatch` allows you to include a regular expression in your redirection criteria, providing many of the benefits of using `RewriteRule`.

A common use for `RewriteRule` is to redirect an entire class of URLs. For example, all URLs in the `/one` directory must be redirected to `http://one.example.com/`, or perhaps all `http` requests must be redirected to `https`.

These situations are better handled by the `Redirect` directive. Remember that `Redirect` preserves path information. That is to say, a redirect for a URL `/one` will also redirect all URLs under that, such as `/one/two.html` and `/one/three/four.html`.

To redirect URLs under `/one` to `http://one.example.com`, do the following:

```
Redirect /one/ http://one.example.com/
```

To redirect `http` URLs to `https`, do the following:

```
<VirtualHost *:80> ServerName www.example.com
Redirect / https://www.example.com/
</VirtualHost >
<VirtualHost *:443> ServerName www.example.com

# ... SSL configuration goes here
</VirtualHost >
```

The use of `RewriteRule` to perform this task may be appropriate if there are other `RewriteRule` directives in the same scope. This is

because, when there are `Redirect` and `RewriteRule` directives in the same scope, the `RewriteRule` directives will run first, regardless of the order of appearance in the configuration file.

In the case of the *http-to-https* redirection, the use of `RewriteRule` would be appropriate if you don't have access to the main server configuration file, and are obliged to perform this task in a `.htaccess` file instead.



The [Alias](#) directive provides mapping from a URI to a directory - usually a directory outside of your [DocumentRoot](#). Although it is possible to perform this mapping with `mod_rewrite`, `Alias` is the preferred method, for reasons of simplicity and performance.

Using Alias

```
Alias /cats /var/www/virtualhosts/felines/htdocs
```

The use of `mod_rewrite` to perform this mapping may be appropriate when you do not have access to the server configuration files. `Alias` may only be used in server or virtualhost context, and not in a `.htaccess` file.

Symbolic links would be another way to accomplish the same thing, if you have `Options FollowSymLinks` enabled on your server.



Virtual Hosting

Although it is possible to handle [virtual hosts with mod_rewrite](#), it is seldom the right way. Creating individual `<VirtualHost>` blocks is almost always the right way to go. In the event that you have an enormous number of virtual hosts, consider using [mod_vhost_alias](#) to create these hosts automatically.

Third-party modules such as [mod_macro](#) are also useful for creating a large number of virtual hosts dynamically.

Using [mod_rewrite](#) for virtualhost creation may be appropriate if you are using a hosting service that does not provide you access to the server configuration files, and you are therefore restricted to configuration using `.htaccess` files.

See the [virtual hosts with mod_rewrite](#) document for more details on how you might accomplish this if it still seems like the right approach.



Simple Proxying

RewriteRule provides the [\[P\]](#) flag to pass rewritten URIs through [mod_proxy](#).

```
RewriteRule ^/?images(.*) http://imageserver.local/images$1 [P]
```

However, in many cases, when there is no actual pattern matching needed, as in the example shown above, the [ProxyPass](#) directive is a better choice. The example here could be rendered as:

```
ProxyPass /images/ http://imageserver.local/images/
```

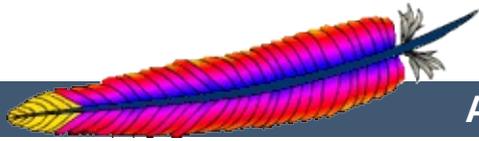
Note that whether you use [RewriteRule](#) or [ProxyPass](#), you'll still need to use the [ProxyPassReverse](#) directive to catch redirects issued from the back-end server:

```
ProxyPassReverse /images/ http://imageserver.local/images/
```

You may need to use RewriteRule instead when there are other RewriteRules in effect in the same scope, as a RewriteRule will usually take effect before a ProxyPass, and so may preempt what you're trying to accomplish.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

RewriteRule Flags

This document discusses the flags which are available to the [RewriteRule](#) directive, providing detailed explanations and examples.

See also

[Module documentation](#)

[mod_rewrite introduction](#)

[Redirection and remapping](#)

[Controlling access](#)

[Virtual hosts](#)

[Proxying](#)

[Using RewriteMap](#)

[Advanced techniques](#)

[When not to use mod_rewrite](#)



A [RewriteRule](#) can have its behavior modified by one or more flags. Flags are included in square brackets at the end of the rule, and multiple flags are separated by commas.

```
RewriteRule pattern target [Flag1,Flag2,Flag3]
```

Each flag (with a few exceptions) has a short form, such as `C0`, as well as a longer form, such as `cookie`. While it is most common to use the short form, it is recommended that you familiarize yourself with the long form, so that you remember what each flag is supposed to do. Some flags take one or more arguments. Flags are not case sensitive.

Flags that alter metadata associated with the request (`T=`, `H=`, `E=`) have no affect in per-directory and `htaccess` context, when a substitution (other than '-') is performed during the same round of rewrite processing.

Presented here are each of the available flags, along with an example of how you might use them.



The [B] flag instructs [RewriteRule](#) to escape non-alphanumeric characters before applying the transformation.

`mod_rewrite` has to unescape URLs before mapping them, so backreferences will be unescaped at the time they are applied. Using the B flag, non-alphanumeric characters in backreferences will be escaped. For example, consider the rule:

```
RewriteRule ^search/(.*)$ /search.php?term=$1
```

Given a search term of 'x & y/z', a browser will encode it as 'x%20%26%20y%2Fz', making the request 'search/x%20%26%20y%2Fz'. Without the B flag, this rewrite rule will map to 'search.php?term=x & y/z', which isn't a valid URL, and so would be encoded as `search.php?term=x%20&y%2Fz=`, which is not what was intended.

With the B flag set on this same rule, the parameters are re-encoded before being passed on to the output URL, resulting in a correct mapping to `/search.php?term=x%20%26%20y%2Fz`.

Note that you may also need to set [AllowEncodedSlashes](#) to On to get this particular example to work, as `httpd` does not allow encoded slashes in URLs, and returns a 404 if it sees one.

This escaping is particularly necessary in a proxy situation, when the backend may break if presented with an unescaped URL.



The [C] or [chain] flag indicates that the [RewriteRule](#) is chained to the next rule. That is, if the rule matches, then it is processed as usual and control moves on to the next rule. However, if it does not match, then the next rule, and any other rules that are chained together, will be skipped.



The [CO], or [cookie] flag, allows you to set a cookie when a particular [RewriteRule](#) matches. The argument consists of three required fields and four optional fields.

The full syntax for the flag, including all attributes, is as follows:

```
[CO=NAME:VALUE:DOMAIN:lifetime:path:secure:httponly]
```

You must declare a name, a value, and a domain for the cookie to be set.

Domain

The domain for which you want the cookie to be valid. This may be a hostname, such as `www.example.com`, or it may be a domain, such as `.example.com`. It must be at least two parts separated by a dot. That is, it may not be merely `.com` or `.net`. Cookies of that kind are forbidden by the cookie security model.

You may optionally also set the following values:

Lifetime

The time for which the cookie will persist, in minutes. A value of 0 indicates that the cookie will persist only for the current browser session. This is the default value if none is specified. Prior to 2.2.28, a value of 0 indicates immediate expiration and it was not possible to specify session lifetime if any later parameters (Path, Secure, httponly) were specified

Path

The path, on the current website, for which the cookie is valid, such as `/customers/` or `/files/download/`.

By default, this is set to `/` - that is, the entire website.

Secure

If set to `secure`, `true`, or `1`, the cookie will only be permitted to be translated via secure (https) connections.

httponly

If set to `HttpOnly`, `true`, or `1`, the cookie will have the `HttpOnly` flag set, which means that the cookie will be inaccessible to JavaScript code on browsers that support this feature.

Several examples are offered here:

```
RewriteEngine On  
RewriteRule ^/index\.html - [CO=frontdoor:yes:.example.com:1440:/]
```

In the example given, the rule doesn't rewrite the request. The "-" rewrite target tells `mod_rewrite` to pass the request through unchanged. Instead, it sets a cookie called 'frontdoor' to a value of 'yes'. The cookie is valid for any host in the `.example.com` domain. It will be set to expire in 1440 minutes (24 hours) and will be returned for all URIs.



The DPI flag causes the PATH_INFO portion of the rewritten URI to be discarded.

This flag is available in version 2.2.12 and later.

In per-directory context, the URI each RewriteRule compares against is the concatenation of the current values of the URI and PATH_INFO.

The current URI can be the initial URI as requested by the client, the result of a previous round of mod_rewrite processing, or the result of a prior rule in the current round of mod_rewrite processing.

In contrast, the PATH_INFO that is appended to the URI before each rule reflects only the value of PATH_INFO before this round of mod_rewrite processing. As a consequence, if large portions of the URI are matched and copied into a substitution in multiple RewriteRule directives, without regard for which parts of the URI came from the current PATH_INFO, the final URI may have multiple copies of PATH_INFO appended to it.

Use this flag on any substitution where the PATH_INFO that resulted from the previous mapping of this request to the filesystem is not of interest. This flag permanently forgets the PATH_INFO established before this round of mod_rewrite processing began. PATH_INFO will not be recalculated until the current round of mod_rewrite processing completes. Subsequent rules during this round of processing will see only the direct result of substitutions, without any PATH_INFO appended.



With the [E], or [env] flag, you can set the value of an environment variable. Note that some environment variables may be set after the rule is run, thus unsetting what you have set. See [the Environment Variables document](#) for more details on how Environment variables work.

The full syntax for this flag is:

```
[E=VAR:VAL] [E=!VAR]
```

VAL may contain backreferences (\$N or %N) which will be expanded.

Using the short form

```
[E=VAR]
```

you can set the environment variable named VAR to an empty value.

The form

```
[E=!VAR]
```

allows to unset a previously set environment variable named VAR.

Environment variables can then be used in a variety of contexts, including CGI programs, other RewriteRule directives, or CustomLog directives.

The following example sets an environment variable called 'image' to a value of '1' if the requested URI is an image file. Then, that environment variable is used to exclude those requests from the access log.

```
RewriteRule \.(png|gif|jpg) - [E=image:1]  
CustomLog logs/access_log combined env=!image
```

Note that this same effect can be obtained using [SetEnvIf](#). This technique is offered as an example, not as a recommendation.



Forbidden

Using the [F] flag causes the server to return a 403 Forbidden status code to the client. While the same behavior can be accomplished using the [Deny](#) directive, this allows more flexibility in assigning a Forbidden status.

The following rule will forbid .exe files from being downloaded from your server.

```
RewriteRule \.exe - [F]
```

This example uses the "-" syntax for the rewrite target, which means that the requested URI is not modified. There's no reason to rewrite to another URI, if you're going to forbid the request.

When using [F], an [L] is implied - that is, the response is returned immediately, and no further rules are evaluated.



The [G] flag forces the server to return a 410 Gone status with the response. This indicates that a resource used to be available, but is no longer available.

As with the [F] flag, you will typically use the "-" syntax for the rewrite target when using the [G] flag:

```
RewriteRule oldproduct - [G,NC]
```

When using [G], an [L] is implied - that is, the response is returned immediately, and no further rules are evaluated.



Forces the resulting request to be handled with the specified handler. For example, one might use this to force all files without a file extension to be parsed by the php handler:

```
RewriteRule !\.\. - [H=application/x-httpd-php]
```

The regular expression above - !\.\. - will match any request that does not contain the literal \. character.

This can be also used to force the handler based on some conditions. For example, the following snippet used in per-server context allows .php files to be *displayed* by mod_php if they are requested with the .phps extension:

```
RewriteRule ^(/source/.\.php)s$ $1 [H=application/x-httpd-php-source]
```

The regular expression above - ^(/source/.\.php)s\$ - will match any request that starts with /source/ followed by 1 or n characters followed by .phps literally. The backreference \$1 refers to the captured match within parenthesis of the regular expression.



The [L] flag causes `mod_rewrite` to stop processing the rule set. In most contexts, this means that if the rule matches, no further rules will be processed. This corresponds to the `last` command in Perl, or the `break` command in C. Use this flag to indicate that the current rule should be applied immediately without considering further rules.

If you are using `RewriteRule` in either `.htaccess` files or in `<Directory>` sections, it is important to have some understanding of how the rules are processed. The simplified form of this is that once the rules have been processed, the rewritten request is handed back to the URL parsing engine to do what it may with it. It is possible that as the rewritten request is handled, the `.htaccess` file or `<Directory>` section may be encountered again, and thus the ruleset may be run again from the start. Most commonly this will happen if one of the rules causes a redirect - either internal or external - causing the request process to start over.

It is therefore important, if you are using `RewriteRule` directives in one of these contexts, that you take explicit steps to avoid rules looping, and not count solely on the [L] flag to terminate execution of a series of rules, as shown below.

The example given here will rewrite any request to `index.php`, giving the original request as a query string argument to `index.php`, however, the `RewriteCond` ensures that if the request is already for `index.php`, the `RewriteRule` will be skipped.

```
RewriteBase /  
RewriteCond %{REQUEST_URI} !=/index.php  
RewriteRule ^(.*) /index.php?req=$1 [L,PT]
```



The [N] flag causes the ruleset to start over again from the top, using the result of the ruleset so far as a starting point. Use with extreme caution, as it may result in loop.

The [Next] flag could be used, for example, if you wished to replace a certain string or letter repeatedly in a request. The example shown here will replace A with B everywhere in a request, and will continue doing so until there are no more As to be replaced.

```
RewriteRule (.*)A(.*) $1B$2 [N]
```

You can think of this as a `while` loop: While this pattern still matches (i.e., while the URI still contains an A), perform this substitution (i.e., replace the A with a B).



Use of the [NC] flag causes the `RewriteRule` to be matched in a case-insensitive manner. That is, it doesn't care whether letters appear as upper-case or lower-case in the matched URI.

In the example below, any request for an image file will be proxied to your dedicated image server. The match is case-insensitive, so that `.jpg` and `.JPG` files are both acceptable, for example.

```
RewriteRule (.*\.(jpg|gif|png))$ http://images.example.com$1 [P,NC]
```



By default, special characters, such as & and ?, for example, will be converted to their hexcode equivalent. Using the [NE] flag prevents that from happening.

```
RewriteRule ^/anchor/(.+) /bigpage.html#$1 [NE,R]
```

The above example will redirect /anchor/xyz to /bigpage.html#xyz. Omitting the [NE] will result in the # being converted to its hexcode equivalent, %23, which will then result in a 404 Not Found error condition.



Use of the [NS] flag prevents the rule from being used on subrequests. For example, a page which is included using an SSI (Server Side Include) is a subrequest, and you may want to avoid rewrites happening on those subrequests. Also, when [mod_dir](#) tries to find out information about possible directory default files (such as `index.html` files), this is an internal subrequest, and you often want to avoid rewrites on such subrequests. On subrequests, it is not always useful, and can even cause errors, if the complete set of rules are applied. Use this flag to exclude problematic rules.

To decide whether or not to use this rule: if you prefix URLs with CGI-scripts, to force them to be processed by the CGI-script, it's likely that you will run into problems (or significant overhead) on sub-requests. In these cases, use this flag.

Images, javascript files, or css files, loaded as part of an HTML page, are not subrequests - the browser requests them as separate HTTP requests.



Use of the [P] flag causes the request to be handled by [mod_proxy](#), and handled via a proxy request. For example, if you wanted all image requests to be handled by a back-end image server, you might do something like the following:

```
RewriteRule /(.*)\.(jpg|gif|png) http://images.example.com/$1.$2 [P]
```

Use of the [P] flag implies [L] - that is, the request is immediately pushed through the proxy, and any following rules will not be considered.

You must make sure that the substitution string is a valid URI (typically starting with `http://hostname`) which can be handled by the [mod_proxy](#). If not, you will get an error from the proxy module. Use this flag to achieve a more powerful implementation of the [ProxyPass](#) directive, to map remote content into the namespace of the local server.

Security Warning

Take care when constructing the target URL of the rule, considering the security impact from allowing the client influence over the set of URLs to which your server will act as a proxy. Ensure that the scheme and hostname part of the URL is either fixed, or does not allow the client undue influence.

Note: [mod_proxy](#) must be enabled in order to use this flag.



The target (or substitution string) in a RewriteRule is assumed to be a file path, by default. The use of the [PT] flag causes it to be treated as a URI instead. That is to say, the use of the [PT] flag causes the result of the [RewriteRule](#) to be passed back through URL mapping, so that location-based mappings, such as [Alias](#), [Redirect](#), or [ScriptAlias](#), for example, might have a chance to take effect.

If, for example, you have an [Alias](#) for /icons, and have a [RewriteRule](#) pointing there, you should use the [PT] flag to ensure that the [Alias](#) is evaluated.

```
Alias /icons /usr/local/apache/icons
RewriteRule /pics/(.+)\.jpg /icons/$1.gif [PT]
```

Omission of the [PT] flag in this case will cause the Alias to be ignored, resulting in a 'File not found' error being returned.

The PT flag implies the L flag: rewriting will be stopped in order to pass the request to the next phase of processing.

Note that the PT flag is implied in per-directory contexts such as [<Directory>](#) sections or in .htaccess files. The only way to circumvent that is to rewrite to -.



When the replacement URI contains a query string, the default behavior of `RewriteRule` is to discard the existing query string, and replace it with the newly generated one. Using the `[QSA]` flag causes the query strings to be combined.

Consider the following rule:

```
RewriteRule /pages/(.+) /page.php?page=$1 [QSA]
```

With the `[QSA]` flag, a request for `/pages/123?one=two` will be mapped to `/page.php?page=123&one=two`. Without the `[QSA]` flag, that same request will be mapped to `/page.php?page=123` - that is, the existing query string will be discarded.



Use of the [R] flag causes a HTTP redirect to be issued to the browser. If a fully-qualified URL is specified (that is, including `http://servername/`) then a redirect will be issued to that location. Otherwise, the current protocol, servername, and port number will be used to generate the URL sent with the redirect.

Any valid HTTP response status code may be specified, using the syntax [R=305], with a 302 status code being used by default if none is specified. The status code specified need not necessarily be a redirect (3xx) status code. However, if a status code is outside the redirect range (300-399) then the substitution string is dropped entirely, and rewriting is stopped as if the L were used.

In addition to response status codes, you may also specify redirect status using their symbolic names: temp (default), permanent, or seeother.

You will almost always want to use [R] in conjunction with [L] (that is, use [R,L]) because on its own, the [R] flag prepends `http://thishost[:thisport]` to the URI, but then passes this on to the next rule in the ruleset, which can often result in 'Invalid URI in request' warnings.



The [S] flag is used to skip rules that you don't want to run. The syntax of the skip flag is [S=N], where N signifies the number of rules to skip (provided the [RewriteRule](#) matches). This can be thought of as a goto statement in your rewrite ruleset. In the following example, we only want to run the [RewriteRule](#) if the requested URI doesn't correspond with an actual file.

```
# Is the request for a non-existent file?
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# If so, skip these two RewriteRules
RewriteRule .? - [S=2]
RewriteRule (.*\gif) images.php?$1
RewriteRule (.*\html) docs.php?$1
```

This technique is useful because a [RewriteCond](#) only applies to the [RewriteRule](#) immediately following it. Thus, if you want to make a [RewriteCond](#) apply to several [RewriteRules](#), one possible technique is to negate those conditions and add a [RewriteRule](#) with a [Skip] flag. You can use this to make pseudo if-then-else constructs: The last rule of the then-clause becomes skip=N, where N is the number of rules in the else-clause:

```
# Does the file exist?
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# Create an if-then-else construct by skipping 3 lines if we meant to
go to the "else" stanza.
RewriteRule .? - [S=3]

# IF the file exists, then:
  RewriteRule (.*\gif) images.php?$1
  RewriteRule (.*\html) docs.php?$1
  # Skip past the "else" stanza.
  RewriteRule .? - [S=1]
# ELSE...
  RewriteRule (.*) 404.php?file=$1
# END
```



Sets the MIME type with which the resulting response will be sent. This has the same effect as the [AddType](#) directive.

For example, you might use the following technique to serve Perl source code as plain text, if requested in a particular way:

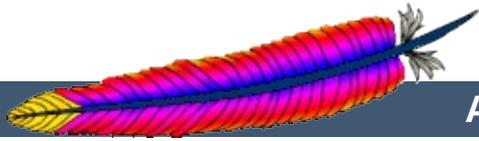
```
# Serve .pl files as plain text
RewriteRule /\.pl$ - [T=text/plain]
```

Or, perhaps, if you have a camera that produces jpeg images without file extensions, you could force those images to be served with the correct MIME type by virtue of their file names:

```
# Files with 'IMG' in the name are jpg images.
RewriteRule IMG - [T=image/jpeg]
```

Please note that this is a trivial example, and could be better done using [<FilesMatch>](#) instead. Always consider the alternate solutions to a problem before resorting to rewrite, which will invariably be a less efficient solution than the alternatives.

If used in per-directory context, use only - (dash) as the substitution *for the entire round of mod_rewrite processing*, otherwise the MIME-type set with this flag is lost due to an internal re-processing (including subsequent rounds of mod_rewrite processing). The L flag can be useful in this context to end the *current* round of mod_rewrite processing.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Rewrite](#)

Apache mod_rewrite Technical Details

This document discusses some of the technical details of mod_rewrite and URL matching.

See also

[Module documentation](#)

[mod_rewrite introduction](#)

[Redirection and remapping](#)

[Controlling access](#)

[Virtual hosts](#)

[Proxying](#)

[Using RewriteMap](#)

[Advanced techniques and tricks](#)

[When not to use mod_rewrite](#)



Internal Processing

The internal processing of this module is very complex but needs to be explained once even to the average user to avoid common mistakes and to let you exploit its full functionality.



First you have to understand that when Apache processes a HTTP request it does this in phases. A hook for each of these phases is provided by the Apache API. `mod_rewrite` uses two of these hooks: the URL-to-filename translation hook which is used after the HTTP request has been read but before any authorization starts and the Fixup hook which is triggered after the authorization phases and after the per-directory config files (`.htaccess`) have been read, but before the content handler is activated.

So, after a request comes in and Apache has determined the corresponding server (or virtual server) the rewriting engine starts processing of all `mod_rewrite` directives from the per-server configuration in the URL-to-filename phase. A few steps later when the final data directories are found, the per-directory configuration directives of `mod_rewrite` are triggered in the Fixup phase. In both situations `mod_rewrite` rewrites URLs either to new URLs or to filenames, although there is no obvious distinction between them. This is a usage of the API which was not intended to be this way when the API was designed, but as of Apache 1.x this is the only way `mod_rewrite` can operate. To make this point more clear remember the following two points:

1. Although `mod_rewrite` rewrites URLs to URLs, URLs to filenames and even filenames to filenames, the API currently provides only a URL-to-filename hook. In Apache 2.0 the two missing hooks will be added to make the processing more clear. But this point has no drawbacks for the user, it is just a fact which should be remembered: Apache does more in the URL-to-filename hook than the API intends for it.
2. Unbelievably `mod_rewrite` provides URL manipulations in per-directory context, *i.e.*, within `.htaccess` files, although these are reached a very long time after the URLs have been translated to filenames. It has to be this way because `.htaccess` files live in the filesystem, so processing has already reached this stage. In other

words: According to the API phases at this time it is too late for any URL manipulations. To overcome this chicken and egg problem `mod_rewrite` uses a trick: When you manipulate a URL/filename in per-directory context `mod_rewrite` first rewrites the filename back to its corresponding URL (which is usually impossible, but see the `RewriteBase` directive below for the trick to achieve this) and then initiates a new internal sub-request with the new URL. This restarts processing of the API phases.

Again `mod_rewrite` tries hard to make this complicated step totally transparent to the user, but you should remember here: While URL manipulations in per-server context are really fast and efficient, per-directory rewrites are slow and inefficient due to this chicken and egg problem. But on the other hand this is the only way `mod_rewrite` can provide (locally restricted) URL manipulations to the average user.

Don't forget these two points!



URL Rewriting

Now when `mod_rewrite` is triggered in these two API phases, it reads the configured rulesets from its configuration structure (which itself was either created on startup for per-server context or during the directory walk of the Apache kernel for per-directory context). Then the URL rewriting engine is started with the contained ruleset (one or more rules together with their conditions). The operation of the URL rewriting engine itself is exactly the same for both configuration contexts. Only the final result processing is different.

The order of rules in the ruleset is important because the rewriting engine processes them in a special (and not very obvious) order. The rule is this: The rewriting engine loops through the ruleset rule by rule (`RewriteRule` directives) and when a particular rule matches it optionally loops through existing corresponding conditions (`RewriteCond` directives). For historical reasons the conditions are given first, and so the control flow is a little bit long-winded. See Figure 1 for more details.

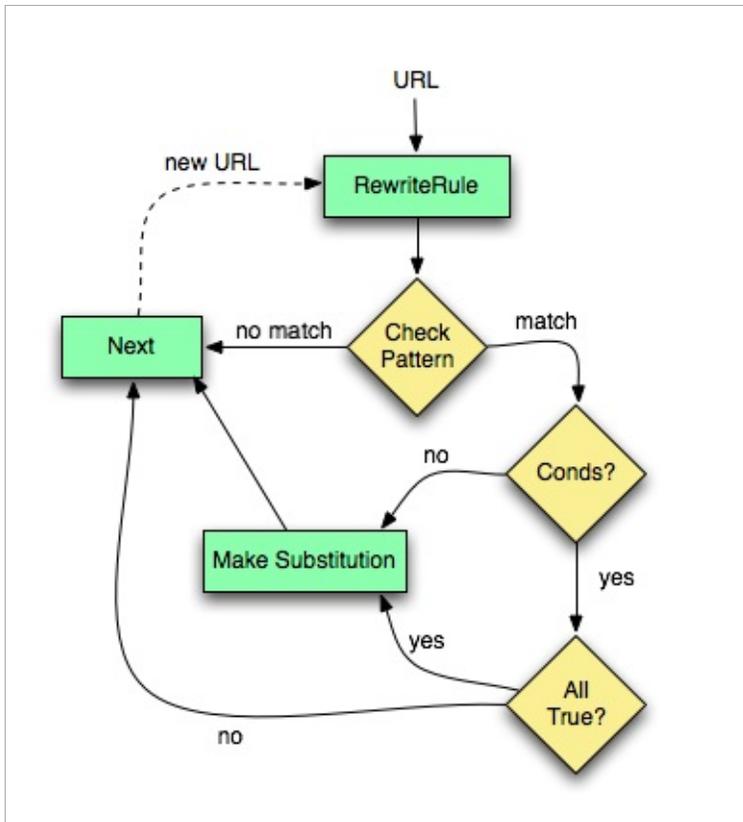


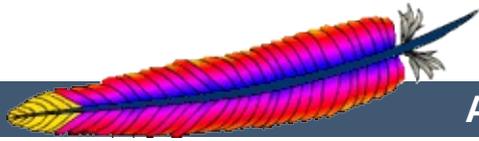
Figure 1: The control

flow through the rewriting ruleset

As you can see, first the URL is matched against the *Pattern* of each rule. When it fails `mod_rewrite` immediately stops processing this rule and continues with the next rule. If the *Pattern* matches, `mod_rewrite` looks for corresponding rule conditions. If none are present, it just substitutes the URL with a new value which is constructed from the string *Substitution* and goes on with its rule-looping. But if conditions exist, it starts an inner loop for processing them in the order that they are listed. For conditions the logic is different: we don't match a pattern against the current URL. Instead we first create a string *TestString* by expanding variables, back-references, map lookups, *etc.* and then we try to match *CondPattern* against it. If the pattern doesn't match, the complete set of conditions and the corresponding rule fails. If the pattern matches, then the next condition is processed until no more conditions are available. If all conditions match, processing is continued with the substitution of the URL with *Substitution*.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [SSL/TLS](#)

SSL/TLS Strong Encryption: An Introduction

The nice thing about standards is that there are so many to choose from. And if you really don't like all the standards you just have to wait another year until the one arises you are looking for.

-- A. Tanenbaum, "Introduction to Computer Networks"

As an introduction this chapter is aimed at readers who are familiar with the Web, HTTP, and Apache, but are not security experts. It is not intended to be a definitive guide to the SSL protocol, nor does it discuss specific techniques for managing certificates in an organization, or the important legal issues of patents and import and export restrictions. Rather, it is intended to provide a common background to `mod_ssl` users by pulling together various concepts, definitions, and examples as a starting point for further exploration.

The presented content is mainly derived, with the author's permission, from the article [Introducing SSL and Certificates using SSLey](#) by [Frederick J. Hirsch](#), of The Open Group Research Institute, which was published in [Web Security: A Matter of Trust](#), World Wide Web Journal, Volume 2, Issue 3, Summer 1997. Please send any positive feedback to [Frederick Hirsch](#) (the original article author) and all negative feedback to [Ralf S. Engelschall](#) (the `mod_ssl` author).



Cryptographic Techniques

Understanding SSL requires an understanding of cryptographic algorithms, message digest functions (aka. one-way or hash functions), and digital signatures. These techniques are the subject of entire books (see for instance [\[AC96\]](#)) and provide the basis for privacy, integrity, and authentication.

Cryptographic Algorithms

Suppose Alice wants to send a message to her bank to transfer some money. Alice would like the message to be private, since it will include information such as her account number and transfer amount. One solution is to use a cryptographic algorithm, a technique that would transform her message into an encrypted form, unreadable until it is decrypted. Once in this form, the message can only be decrypted by using a secret key. Without the key the message is useless: good cryptographic algorithms make it so difficult for intruders to decode the original text that it isn't worth their effort.

There are two categories of cryptographic algorithms: conventional and public key.

Conventional cryptography

also known as symmetric cryptography, requires the sender and receiver to share a key: a secret piece of information that may be used to encrypt or decrypt a message. As long as this key is kept secret, nobody other than the sender or recipient can read the message. If Alice and the bank know a secret key, then they can send each other private messages. The task of sharing a key between sender and recipient before communicating, while also keeping it secret from others, can be problematic.

Public key cryptography

also known as asymmetric cryptography, solves the key exchange problem by defining an algorithm which uses two keys, each of which may be used to encrypt a message. If one key is used to

encrypt a message then the other must be used to decrypt it. This makes it possible to receive secure messages by simply publishing one key (the public key) and keeping the other secret (the private key).

Anyone can encrypt a message using the public key, but only the owner of the private key will be able to read it. In this way, Alice can send private messages to the owner of a key-pair (the bank), by encrypting them using their public key. Only the bank will be able to decrypt them.

Message Digests

Although Alice may encrypt her message to make it private, there is still a concern that someone might modify her original message or substitute it with a different one, in order to transfer the money to themselves, for instance. One way of guaranteeing the integrity of Alice's message is for her to create a concise summary of her message and send this to the bank as well. Upon receipt of the message, the bank creates its own summary and compares it with the one Alice sent. If the summaries are the same then the message has been received intact.

A summary such as this is called a *message digest*, *one-way function* or *hash function*. Message digests are used to create a short, fixed-length representation of a longer, variable-length message. Digest algorithms are designed to produce a unique digest for each message. Message digests are designed to make it impractically difficult to determine the message from the digest and (in theory) impossible to find two different messages which create the same digest -- thus eliminating the possibility of substituting one message for another while maintaining the same digest.

Another challenge that Alice faces is finding a way to send the digest to the bank securely; if the digest is not sent securely, its integrity may be compromised and with it the possibility for the bank to determine the integrity of the original message. Only if the digest is sent securely can

the integrity of the associated message be determined.

One way to send the digest securely is to include it in a digital signature.

Digital Signatures

When Alice sends a message to the bank, the bank needs to ensure that the message is really from her, so an intruder cannot request a transaction involving her account. A *digital signature*, created by Alice and included with the message, serves this purpose.

Digital signatures are created by encrypting a digest of the message and other information (such as a sequence number) with the sender's private key. Though anyone can *decrypt* the signature using the public key, only the sender knows the private key. This means that only the sender can have signed the message. Including the digest in the signature means the signature is only good for that message; it also ensures the integrity of the message since no one can change the digest and still sign it.

To guard against interception and reuse of the signature by an intruder at a later date, the signature contains a unique sequence number. This protects the bank from a fraudulent claim from Alice that she did not send the message -- only she could have signed it (non-repudiation).



Although Alice could have sent a private message to the bank, signed it and ensured the integrity of the message, she still needs to be sure that she is really communicating with the bank. This means that she needs to be sure that the public key she is using is part of the bank's key-pair, and not an intruder's. Similarly, the bank needs to verify that the message signature really was signed by the private key that belongs to Alice.

If each party has a certificate which validates the other's identity, confirms the public key and is signed by a trusted agency, then both can be assured that they are communicating with whom they think they are. Such a trusted agency is called a *Certificate Authority* and certificates are used for authentication.

Certificate Contents

A certificate associates a public key with the real identity of an individual, server, or other entity, known as the subject. As shown in [Table 1](#), information about the subject includes identifying information (the distinguished name) and the public key. It also includes the identification and signature of the Certificate Authority that issued the certificate and the period of time during which the certificate is valid. It may have additional information (or extensions) as well as administrative information for the Certificate Authority's use, such as a serial number.

Table 1: Certificate Information

Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of Validity	Not Before Date, Not After Date
Administrative Information	Version, Serial Number
Extended Information	Basic Constraints, Netscape Flags, etc.

A distinguished name is used to provide an identity in a specific context -- for instance, an individual might have a personal certificate as well as

one for their identity as an employee. Distinguished names are defined by the X.509 standard [X509], which defines the fields, field names and abbreviations used to refer to the fields (see [Table 2](#)).

Table 2: Distinguished Name Information

DN Field	Abbrev.	Description	Example
Common Name	CN	Name being certified	CN=Joe Average
Organization or Company	O	Name is associated with this organization	O=Snake Oil, Ltd.
Organizational Unit	OU	Name is associated with this organization unit, such as a department	OU=Research Institute
City/Locality	L	Name is located in this City	L=Snake City
State/Province	ST	Name is located in this State/Province	ST=Desert
Country	C	Name is located in this Country (ISO code)	C=XZ

A Certificate Authority may define a policy specifying which distinguished field names are optional and which are required. It may also place requirements upon the field contents, as may users of certificates. For example, a Netscape browser requires that the Common Name for a certificate representing a server matches a wildcard pattern for the domain name of that server, such as `*.snakeoil.com`.

The binary format of a certificate is defined using the ASN.1 notation [X208] [PKCS]. This notation defines how to specify the contents and encoding rules define how this information is translated into binary form. The binary encoding of the certificate is defined using Distinguished Encoding Rules (DER), which are based on the more general Basic

Encoding Rules (BER). For those transmissions which cannot handle binary, the binary form may be translated into an ASCII form by using Base64 encoding [[MIME](#)]. When placed between begin and end delimiter lines (as below), this encoded version is called a PEM ("Privacy Enhanced Mail") encoded certificate.

Example of a PEM-encoded certificate (snakeoil.crt)

```
-----BEGIN CERTIFICATE-----
MIIC7jCCAlegAwIBAgIBATANBgkqhkiG9w0BAQQFADCBqTELMAkGA1UEBhMCWFkx
FTATBgNVBAGTDFNuYWt1IERlc2VydDETMDEGA1UEBxMKU25ha2UgVG93bjEXMBUG
A1UEChMOU25ha2UgT2lsLCBmdGQxHjAcBgNVBAsTFUNlcjZm1jYXR1IEF1dGhv
cm10eTEVMBMGA1UEAxMMU25ha2UgT2lsIENBMR4wHAYJKoZIhvcNAQkBFg9jYUBz
bmFrZW9pbC5kb20wHhcNOTGxMDIxMDg1ODM2WhcNOTkxMDIxMDg1ODM2WjCBpzEL
MAkGA1UEBhMCWFkxFTATBgNVBAGTDFNuYWt1IERlc2VydDETMDEGA1UEBxMKU25h
a2UgVG93bjEXMBUGA1UEChMOU25ha2UgT2lsLCBmdGQxHjAcBgNVBAsTDld1YnNl
cnZlcjBUZWFtMRkwFwYDVQQDExB3d3cuc25ha2VvaWwuZG9tMR8wHQYJKoZIhvcN
AQkBFhB3d3dAc25ha2VvaWwuZG9tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDH9Ge/s2zcH+da+rPTx/DPRp3xGjHZ4GG6pCmvADIETtBtKBFAcZ64n+Dy7Np8b
vKR+yy5DGQiijsH1D/j8HlGE+q4TZ80Fk7BNBFazHxFbYI40KMiCxdKzdif1yfaa
lWoANf1Az1SdbxeGVHoT0K+gT5w3UxwZKv2DLbCTzLZyPwIDAQABoyYwJDAPBgNV
HRMECDAGAQH/AgEAMBEGCWCsAGG+EIBAQQEAWIAQDANBgkqhkiG9w0BAQQFAA0B
gQAZUIHAL4D09oE6Lv2k56Gp380BDuILvLg1v1KL8mQR+KFjghCrtpqaztZqcDt
2q2QoyulCgSzHbEGmi0EsdKpfg6mp0penssIFePYNI+/8u9HT4LuKMJX15hxBam7
dUHziCxBVC1lnHyYGjDuAMhe396lYAn8bClD1/L4NMGBCQ==
-----END CERTIFICATE-----
```

Certificate Authorities

By verifying the information in a certificate request before granting the certificate, the Certificate Authority assures itself of the identity of the private key owner of a key-pair. For instance, if Alice requests a personal certificate, the Certificate Authority must first make sure that Alice really is the person the certificate request claims she is.

Certificate Chains

A Certificate Authority may also issue a certificate for another Certificate Authority. When examining a certificate, Alice may need to examine the certificate of the issuer, for each parent Certificate Authority, until reaching one which she has confidence in. She may decide to trust only

certificates with a limited chain of issuers, to reduce her risk of a "bad" certificate in the chain.

Creating a Root-Level CA

As noted earlier, each certificate requires an issuer to assert the validity of the identity of the certificate subject, up to the top-level Certificate Authority (CA). This presents a problem: who can vouch for the certificate of the top-level authority, which has no issuer? In this unique case, the certificate is "self-signed", so the issuer of the certificate is the same as the subject. Browsers are preconfigured to trust well-known certificate authorities, but it is important to exercise extra care in trusting a self-signed certificate. The wide publication of a public key by the root authority reduces the risk in trusting this key -- it would be obvious if someone else publicized a key claiming to be the authority.

A number of companies, such as [Thawte](#) and [VeriSign](#) have established themselves as Certificate Authorities. These companies provide the following services:

- Verifying certificate requests
- Processing certificate requests
- Issuing and managing certificates

It is also possible to create your own Certificate Authority. Although risky in the Internet environment, it may be useful within an Intranet where the organization can easily verify the identities of individuals and servers.

Certificate Management

Establishing a Certificate Authority is a responsibility which requires a solid administrative, technical and management framework. Certificate Authorities not only issue certificates, they also manage them -- that is, they determine for how long certificates remain valid, they renew them and keep lists of certificates that were issued in the past but are no longer valid (Certificate Revocation Lists, or CRLs).

For example, if Alice is entitled to a certificate as an employee of a company but has now left that company, her certificate may need to be revoked. Because certificates are only issued after the subject's identity has been verified and can then be passed around to all those with whom the subject may communicate, it is impossible to tell from the certificate alone that it has been revoked. Therefore when examining certificates for validity it is necessary to contact the issuing Certificate Authority to check CRLs -- this is usually not an automated part of the process.

Note

If you use a Certificate Authority that browsers are not configured to trust by default, it is necessary to load the Certificate Authority certificate into the browser, enabling the browser to validate server certificates signed by that Certificate Authority. Doing so may be dangerous, since once loaded, the browser will accept all certificates signed by that Certificate Authority.



The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP). SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy.

The protocol is designed to support a range of choices for specific algorithms used for cryptography, digests and signatures. This allows algorithm selection for specific servers to be made based on legal, export or other concerns and also enables the protocol to take advantage of new algorithms. Choices are negotiated between client and server when establishing a protocol session.

Table 4: Versions of the SSL protocol

Version	Source	Description	Browser Support
SSL v2.0	Vendor Standard (from Netscape Corp.) [SSL2]	First SSL protocol for which implementations exist	- NS Navigator 1.x/2.x - MS IE 3.x - Lynx/2.8+OpenSSL
SSL v3.0	Expired Internet Draft (from Netscape Corp.) [SSL3]	Revisions to prevent specific security attacks, add non-RSA ciphers and support for certificate chains	- NS Navigator 2.x/3.x/4.x - MS IE 3.x/4.x - Lynx/2.8+OpenSSL
TLS v1.0	Proposed Internet Standard (from IETF)	Revision of SSL 3.0 to update the MAC layer to HMAC, add block padding for block ciphers, message order	- Lynx/2.8+OpenSSL

	[TLS1]	standardization and more alert messages.	
--	------------------------	--	--

There are a number of versions of the SSL protocol, as shown in [Table 4](#). As noted there, one of the benefits in SSL 3.0 is that it adds support of certificate chain loading. This feature allows a server to pass a server certificate along with issuer certificates to the browser. Chain loading also permits the browser to validate the server certificate, even if Certificate Authority certificates are not installed for the intermediate issuers, since they are included in the certificate chain. SSL 3.0 is the basis for the Transport Layer Security [\[TLS\]](#) protocol standard, currently in development by the Internet Engineering Task Force (IETF).

Establishing a Session

The SSL session is established by following a handshake sequence between client and server, as shown in [Figure 1](#). This sequence may vary, depending on whether the server is configured to provide a server certificate or request a client certificate. Although cases exist where additional handshake steps are required for management of cipher information, this article summarizes one common scenario. See the SSL specification for the full range of possibilities.

Note

Once an SSL session has been established, it may be reused. This avoids the performance penalty of repeating the many steps needed to start a session. To do this, the server assigns each SSL session a unique session identifier which is cached in the server and which the client can use in future connections to reduce the handshake time (until the session identifier expires from the cache of the server).

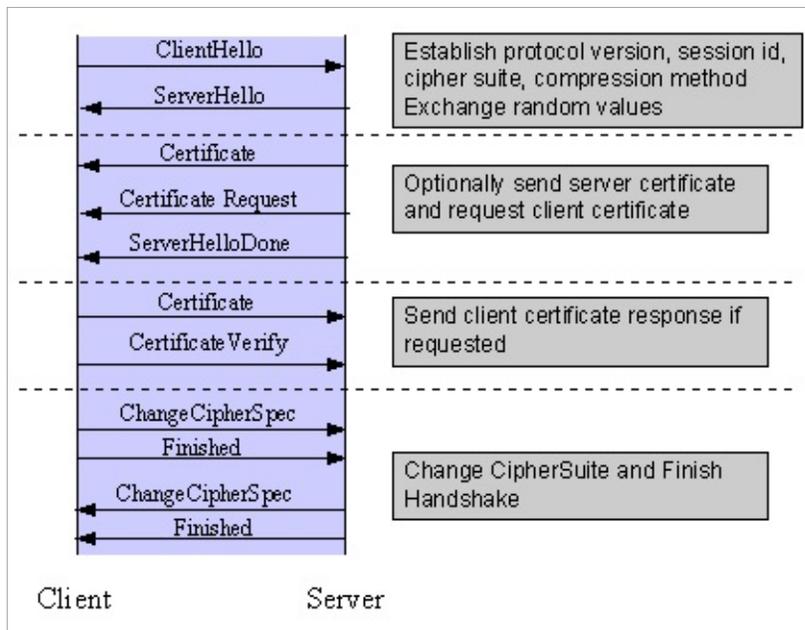


Figure 1: Simplified SSL Handshake Sequence

The elements of the handshake sequence, as used by the client and server, are listed below:

1. Negotiate the Cipher Suite to be used during data transfer
2. Establish and share a session key between client and server
3. Optionally authenticate the server to the client
4. Optionally authenticate the client to the server

The first step, Cipher Suite Negotiation, allows the client and server to choose a Cipher Suite supported by both of them. The SSL3.0 protocol specification defines 31 Cipher Suites. A Cipher Suite is defined by the following components:

- Key Exchange Method
- Cipher for Data Transfer
- Message Digest for creating the Message Authentication Code (MAC)

These three elements are described in the sections that follow.

Key Exchange Method

The key exchange method defines how the shared secret symmetric cryptography key used for application data transfer will be agreed upon by client and server. SSL 2.0 uses RSA key exchange only, while SSL 3.0 supports a choice of key exchange algorithms including RSA key exchange (when certificates are used), and Diffie-Hellman key exchange (for exchanging keys without certificates, or without prior communication between client and server).

One variable in the choice of key exchange methods is digital signatures -- whether or not to use them, and if so, what kind of signatures to use. Signing with a private key provides protection against a man-in-the-middle-attack during the information exchange used to generating the shared key [[AC96](#), p516].

Cipher for Data Transfer

SSL uses conventional symmetric cryptography, as described earlier, for encrypting messages in a session. There are nine choices of how to encrypt, including the option not to encrypt:

- No encryption
- Stream Ciphers
 - RC4 with 40-bit keys
 - RC4 with 128-bit keys
- CBC Block Ciphers
 - RC2 with 40 bit key
 - DES with 40 bit key
 - DES with 56 bit key
 - Triple-DES with 168 bit key
 - Idea (128 bit key)
 - Fortezza (96 bit key)

"CBC" refers to Cipher Block Chaining, which means that a portion of the

previously encrypted cipher text is used in the encryption of the current block. "DES" refers to the Data Encryption Standard [AC96, ch12], which has a number of variants (including DES40 and 3DES_EDE). "Idea" is currently one of the best and cryptographically strongest algorithms available, and "RC2" is a proprietary algorithm from RSA DSI [AC96, ch13].

Digest Function

The choice of digest function determines how a digest is created from a record unit. SSL supports the following:

- No digest (Null choice)
- MD5, a 128-bit hash
- Secure Hash Algorithm (SHA-1), a 160-bit hash

The message digest is used to create a Message Authentication Code (MAC) which is encrypted with the message to verify integrity and to protect against replay attacks.

Handshake Sequence Protocol

The handshake sequence uses three protocols:

- The *SSL Handshake Protocol* for performing the client and server SSL session establishment.
- The *SSL Change Cipher Spec Protocol* for actually establishing agreement on the Cipher Suite for the session.
- The *SSL Alert Protocol* for conveying SSL error messages between client and server.

These protocols, as well as application protocol data, are encapsulated in the *SSL Record Protocol*, as shown in [Figure 2](#). An encapsulated protocol is transferred as data by the lower layer protocol, which does not examine the data. The encapsulated protocol has no knowledge of the underlying protocol.

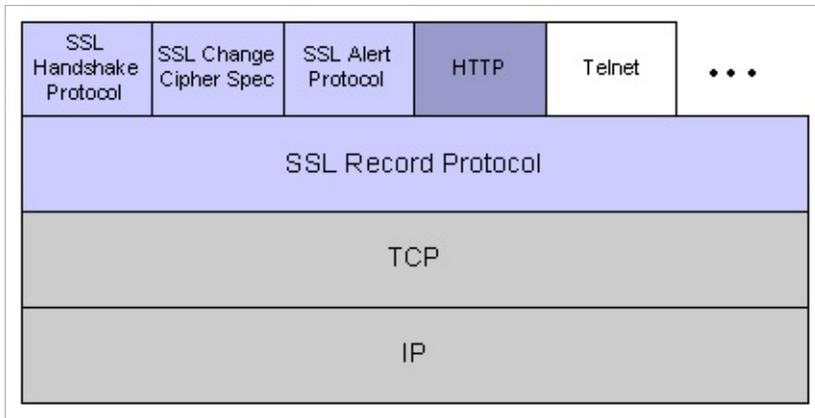


Figure 2: SSL Protocol Stack

The encapsulation of SSL control protocols by the record protocol means that if an active session is renegotiated the control protocols will be transmitted securely. If there was no previous session, the Null cipher suite is used, which means there will be no encryption and messages will have no integrity digests, until the session has been established.

Data Transfer

The SSL Record Protocol, shown in [Figure 3](#), is used to transfer application and SSL Control data between the client and server, where necessary fragmenting this data into smaller units, or combining multiple higher level protocol data messages into single units. It may compress, attach digest signatures, and encrypt these units before transmitting them using the underlying reliable transport protocol (Note: currently, no major SSL implementations include support for compression).

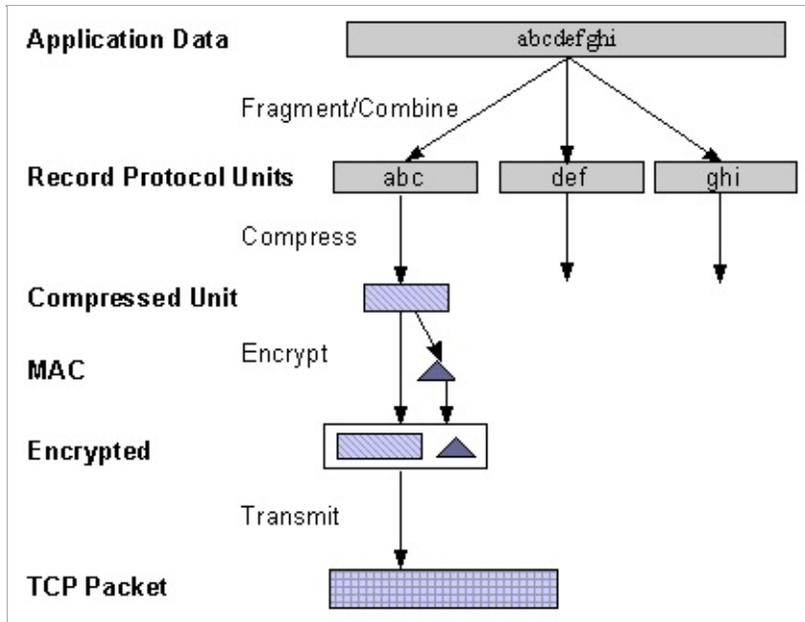


Figure 3: SSL Record Protocol

Securing HTTP Communication

One common use of SSL is to secure Web HTTP communication between a browser and a webserver. This does not preclude the use of non-secured HTTP - the secure version (called HTTPS) is the same as plain HTTP over SSL, but uses the URL scheme `https` rather than `http`, and a different server port (by default, port 443). This functionality is a large part of what [mod_ssl](#) provides for the Apache webserver.



[AC96]

Bruce Schneier, *“Applied Cryptography”*, 2nd Edition, Wiley, 1996. See <http://www.counterpane.com/> for various other materials by Bruce Schneier.

[X208]

ITU-T Recommendation X.208, *“Specification of Abstract Syntax Notation One (ASN.1)”*, 1988. See for instance <http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.208-198811-I>.

[X509]

ITU-T Recommendation X.509, *“The Directory - Authentication Framework”*. See for instance <http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.509>.

[PKCS]

“Public Key Cryptography Standards (PKCS)”, RSA Laboratories Technical Notes, See <http://www.rsasecurity.com/rsalabs/pkcs/>.

[MIME]

N. Freed, N. Borenstein, *“Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies”*, RFC2045. See for instance <http://ietf.org/rfc/rfc2045.txt>.

[SSL2]

Kipp E.B. Hickman, *“The SSL Protocol”*, 1995. See http://www.netscape.com/eng/security/SSL_2.html.

[SSL3]

Alan O. Freier, Philip Karlton, Paul C. Kocher, *“The SSL Protocol Version 3.0”*, 1996. See <http://www.netscape.com/eng/ssl3/draft302.txt>.

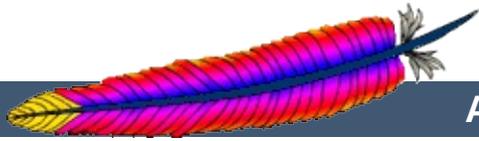
[TLS1]

Tim Dierks, Christopher Allen, *“The TLS Protocol Version 1.0”*, 1999.

See <http://ietf.org/rfc/rfc2246.txt>.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [SSL/TLS](#)

SSL/TLS Strong Encryption: Compatibility

This page covers backwards compatibility between `mod_ssl` and other SSL solutions. `mod_ssl` is not the only SSL solution for Apache; four additional products are (or were) also available: Ben Laurie's freely available [Apache-SSL](#) (from where `mod_ssl` were originally derived in 1998), Red Hat's commercial Secure Web Server (which was based on `mod_ssl`), Covalent's commercial Raven SSL Module (also based on `mod_ssl`) and finally C2Net's (now Red Hat's) commercial product [Stronghold](#) (based on a different evolution branch named Sioux up to Stronghold 2.x and based on `mod_ssl` since Stronghold 3.x).

`mod_ssl` mostly provides a superset of the functionality of all the other solutions, so it's simple to migrate from one of the older modules to `mod_ssl`. The configuration directives and environment variable names used by the older SSL solutions vary from those used in `mod_ssl`; mapping tables are included here to give the equivalents used by `mod_ssl`.



The mapping between configuration directives used by Apache-SSL 1.x and mod_ssl 2.0.x is given in [Table 1](#). The mapping from Sioux 1.x and Stronghold 2.x is only partial because of special functionality in these interfaces which mod_ssl doesn't provide.

Table 1: Configuration Directive Mapping

Old Directive	mod_ssl Directive
Apache-SSL 1.x & mod_ssl 2.0.x compatibility:	
SSLEnable	SSLEngine on
SSLDisable	SSLEngine off
SSLLogFile <i>file</i>	-
SSLRequiredCiphers <i>spec</i>	SSLCipherSuite <i>spec</i>
SSLRequireCipher <i>c1 ...</i>	SSLRequire %{SSL_CIPHER} in {"c1", ...}
SSLBanCipher <i>c1 ...</i>	SSLRequire not (%{SSL_CIPHER} in {"c1", ...})
SSLFakeBasicAuth	SSLOptions +FakeBasicAuth
SSLCacheServerPath <i>dir</i>	-
SSLCacheServerPort <i>integer</i>	-
Apache-SSL 1.x compatibility:	
SSLExportClientCertificates	SSLOptions +ExportCertData
SSLCacheServerRunDir <i>dir</i>	-
Sioux 1.x compatibility:	
SSL_CertFile <i>file</i>	SSLCertificateFile <i>file</i>
SSL_KeyFile <i>file</i>	SSLCertificateKeyFile <i>file</i>
SSL_CipherSuite <i>arg</i>	SSLCipherSuite <i>arg</i>

SSL_X509VerifyDir <i>arg</i>	SSLCACertificatePath <i>arg</i>
SSL_Log <i>file</i>	-
SSL_Connect <i>flag</i>	SSLEngine <i>flag</i>
SSL_ClientAuth <i>arg</i>	SSLVerifyClient <i>arg</i>
SSL_X509VerifyDepth <i>arg</i>	SSLVerifyDepth <i>arg</i>
SSL_FetchKeyPhraseFrom <i>arg</i>	-
SSL_SessionDir <i>dir</i>	-
SSL_Require <i>expr</i>	-
SSL_CertFileType <i>arg</i>	-
SSL_KeyFileType <i>arg</i>	-
SSL_X509VerifyPolicy <i>arg</i>	-
SSL_LogX509Attributes <i>arg</i>	-
Stronghold 2.x compatibility:	
StrongholdAccelerator <i>engine</i>	SSLCryptoDevice <i>engine</i>
StrongholdKey <i>dir</i>	-
StrongholdLicenseFile <i>dir</i>	-
SSLFlag <i>flag</i>	SSLEngine <i>flag</i>
SSLSessionLockFile <i>file</i>	SSLMutex <i>file</i>

SSLCipherList <i>spec</i>	SSLCipherSuite <i>spec</i>
RequireSSL	SSLRequireSSL
SSLLogFile <i>file</i>	-
SSLRoot <i>dir</i>	-
SSL_CertificateLogDir <i>dir</i>	-
AuthCertDir <i>dir</i>	-
SSL_Group <i>name</i>	-
SSLProxyMachineCertPath <i>dir</i>	SSLProxyMachineCertificate <i>dir</i>
SSLProxyMachineCertFile <i>file</i>	SSLProxyMachineCertificate <i>file</i>
SSLProxyCipherList <i>spec</i>	SSLProxyCipherSpec <i>spec</i>



The mapping between environment variable names used by the older SSL solutions and the names used by mod_ssl is given in [Table 2](#).

Table 2: Environment Variable Derivation

Old Variable	mod_ssl Variable	C
SSL_PROTOCOL_VERSION	SSL_PROTOCOL	re
SSLEAY_VERSION	SSL_VERSION_LIBRARY	re
HTTPS_SECRETKEYSIZE	SSL_CIPHER_USEKEYSIZE	re
HTTPS_KEYSIZE	SSL_CIPHER_ALGKEYSIZE	re
HTTPS_CIPHER	SSL_CIPHER	re
HTTPS_EXPORT	SSL_CIPHER_EXPORT	re
SSL_SERVER_KEY_SIZE	SSL_CIPHER_ALGKEYSIZE	re
SSL_SERVER_CERTIFICATE	SSL_SERVER_CERT	re
SSL_SERVER_CERT_START	SSL_SERVER_V_START	re
SSL_SERVER_CERT_END	SSL_SERVER_V_END	re
SSL_SERVER_CERT_SERIAL	SSL_SERVER_M_SERIAL	re
SSL_SERVER_SIGNATURE_ALGORITHM	SSL_SERVER_A_SIG	re
SSL_SERVER_DN	SSL_SERVER_S_DN	re
SSL_SERVER_CN	SSL_SERVER_S_DN_CN	re
SSL_SERVER_EMAIL	SSL_SERVER_S_DN_Email	re
SSL_SERVER_O	SSL_SERVER_S_DN_O	re
SSL_SERVER_OU	SSL_SERVER_S_DN_OU	re
SSL_SERVER_C	SSL_SERVER_S_DN_C	re
SSL_SERVER_SP	SSL_SERVER_S_DN_SP	re
SSL_SERVER_L	SSL_SERVER_S_DN_L	re
SSL_SERVER_IDN	SSL_SERVER_I_DN	re
SSL_SERVER_ICN	SSL_SERVER_I_DN_CN	re
SSL_SERVER_IEMAIL	SSL_SERVER_I_DN_Email	re
SSL_SERVER_IO	SSL_SERVER_I_DN_O	re

SSL_SERVER_IOU	SSL_SERVER_I_DN_OU	re
SSL_SERVER_IC	SSL_SERVER_I_DN_C	re
SSL_SERVER_ISP	SSL_SERVER_I_DN_SP	re
SSL_SERVER_IL	SSL_SERVER_I_DN_L	re
SSL_CLIENT_CERTIFICATE	SSL_CLIENT_CERT	re
SSL_CLIENT_CERT_START	SSL_CLIENT_V_START	re
SSL_CLIENT_CERT_END	SSL_CLIENT_V_END	re
SSL_CLIENT_CERT_SERIAL	SSL_CLIENT_M_SERIAL	re
SSL_CLIENT_SIGNATURE_ALGORITHM	SSL_CLIENT_A_SIG	re
SSL_CLIENT_DN	SSL_CLIENT_S_DN	re
SSL_CLIENT_CN	SSL_CLIENT_S_DN_CN	re
SSL_CLIENT_EMAIL	SSL_CLIENT_S_DN_Email	re
SSL_CLIENT_O	SSL_CLIENT_S_DN_O	re
SSL_CLIENT_OU	SSL_CLIENT_S_DN_OU	re
SSL_CLIENT_C	SSL_CLIENT_S_DN_C	re
SSL_CLIENT_SP	SSL_CLIENT_S_DN_SP	re
SSL_CLIENT_L	SSL_CLIENT_S_DN_L	re
SSL_CLIENT_IDN	SSL_CLIENT_I_DN	re
SSL_CLIENT_ICN	SSL_CLIENT_I_DN_CN	re
SSL_CLIENT_IEMAIL	SSL_CLIENT_I_DN_Email	re
SSL_CLIENT_IO	SSL_CLIENT_I_DN_O	re
SSL_CLIENT_IOU	SSL_CLIENT_I_DN_OU	re
SSL_CLIENT_IC	SSL_CLIENT_I_DN_C	re
SSL_CLIENT_ISP	SSL_CLIENT_I_DN_SP	re
SSL_CLIENT_IL	SSL_CLIENT_I_DN_L	re
SSL_EXPORT	SSL_CIPHER_EXPORT	re
SSL_KEYSIZE	SSL_CIPHER_ALGKEYSIZE	re
SSL_SECKEYSIZE	SSL_CIPHER_USEKEYSIZE	re
SSL_SSLEAY_VERSION	SSL_VERSION_LIBRARY	re
SSL_STRONG_CRYPTO	-	Ni

		sl by m
SSL_SERVER_KEY_EXP	-	sl by m
SSL_SERVER_KEY_ALGORITHM	-	sl by m
SSL_SERVER_KEY_SIZE	-	sl by m
SSL_SERVER_SESSIONDIR	-	sl by m
SSL_SERVER_CERTIFICATELOGDIR	-	sl by m
SSL_SERVER_CERTFILE	-	sl by m
SSL_SERVER_KEYFILE	-	sl by m
SSL_SERVER_KEYFILETYPE	-	sl

		by m
SSL_CLIENT_KEY_EXP	-	Ne su by m
SSL_CLIENT_KEY_ALGORITHM	-	Ne su by m
SSL_CLIENT_KEY_SIZE	-	Ne su by m

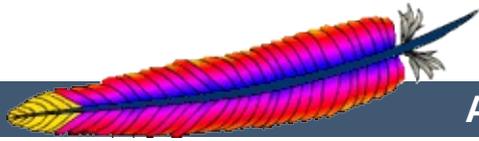


When `mod_ssl` is enabled, additional functions exist for the [Custom Log Format](#) of `mod_log_config` as documented in the Reference Chapter. Beside the ``%{varname}x`" eXtension format function which can be used to expand any variables provided by any module, an additional Cryptography ``%{name}c`" cryptography format function exists for backward compatibility. The currently implemented function calls are listed in [Table 3](#).

Table 3: Custom Log Cryptography Function

Function Call	Description
<code>%...{version}c</code>	SSL protocol version
<code>%...{cipher}c</code>	SSL cipher
<code>%...{subjectdn}c</code>	Client Certificate Subject Distinguished Name
<code>%...{issuerdn}c</code>	Client Certificate Issuer Distinguished Name
<code>%...{errcode}c</code>	Certificate Verification Error (numerical)
<code>%...{errstr}c</code>	Certificate Verification Error (string)

Copyright 2017 The Apache Software Foundation.
 Licensed under the [Apache License, Version 2.0](#).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [SSL/TLS](#)

SSL/TLS Strong Encryption: How-To

The solution to this problem is trivial and is left as an exercise for the reader.

-- Standard textbook cookie

How to solve particular security problems for an SSL-aware webserver is not always obvious because of the interactions between SSL, HTTP and Apache's way of processing requests. This chapter gives instructions on how to solve some typical situations. Treat it as a first step to find out the final solution, but always try to understand the stuff before you use it. Nothing is worse than using a security solution without knowing its restrictions and how it interacts with other systems.



- [How can I create a real SSLv2-only server?](#)
- [How can I create an SSL server which accepts strong encryption only?](#)
- [How can I create an SSL server which accepts strong encryption only, but allows export browsers to upgrade to stronger encryption?](#)
- [How can I create an SSL server which accepts all types of ciphers in general, but requires a strong cipher for access to a particular URL?](#)

How can I create a real SSLv2-only server?

The following creates an SSL server which speaks only the SSLv2 protocol and its ciphers.

httpd.conf

```
SSLProtocol -all +SSLv2
SSLCipherSuite SSLv2:+HIGH:+MEDIUM:+LOW:+EXP
```

How can I create an SSL server which accepts strong encryption only?

The following enables only the seven strongest ciphers:

httpd.conf

```
SSLProtocol all
SSLCipherSuite HIGH:MEDIUM
```

How can I create an SSL server which accepts strong encryption only, but allows export browsers to upgrade to stronger encryption?

This facility is called Server Gated Cryptography (SGC) and requires a Global ID server certificate, signed by a special CA certificate from Verisign. This enables strong encryption in 'export' versions of browsers, which traditionally could not support it (because of US export

restrictions).

When a browser connects with an export cipher, the server sends its Global ID certificate. The browser verifies this, and can then upgrade its cipher suite before any HTTP communication takes place. The problem lies in allowing browsers to upgrade in this fashion, but still requiring strong encryption. In other words, we want browsers to either start a connection with strong encryption, or to start with export ciphers but upgrade to strong encryption before beginning HTTP communication.

This can be done as follows:

httpd.conf

```
# allow all ciphers for the initial handshake,
# so export browsers can upgrade via SGC facility
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

<Directory /usr/local/apache2/htdocs>
# but finally deny all browsers which haven't upgraded
SSLRequire %{SSL_CIPHER_USEKEYSIZE} >= 128
</Directory>
```

How can I create an SSL server which accepts all types of ciphers in general, but requires a strong ciphers for access to a particular URL?

Obviously, a server-wide [SSLCipherSuite](#) which restricts ciphers to the strong variants, isn't the answer here. However, [mod_ssl](#) can be reconfigured within `Location` blocks, to give a per-directory solution, and can automatically force a renegotiation of the SSL parameters to meet the new configuration. This can be done as follows:

```
# be liberal in general
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

<Location /strong/area>
# but https://hostname/strong/area/ and below
# requires strong ciphers
```

SSLCipherSuite HIGH:MEDIUM
</Location>



- [How can I force clients to authenticate using certificates?](#)
- [How can I force clients to authenticate using certificates for a particular URL, but still allow arbitrary clients to access the rest of the server?](#)
- [How can I allow only clients who have certificates to access a particular URL, but allow all clients to access the rest of the server?](#)
- [How can I require HTTPS with strong ciphers, and either basic authentication or client certificates, for access to part of the Intranet website, for clients coming from the Internet?](#)

How can I force clients to authenticate using certificates?

When you know all of your users (eg, as is often the case on a corporate Intranet), you can require plain certificate authentication. All you need to do is to create client certificates signed by your own CA certificate (ca.crt) and then verify the clients against this certificate.

httpd.conf

```
# require a client certificate which has to be directly
# signed by our CA certificate in ca.crt
SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile conf/ssl.crt/ca.crt
```

How can I force clients to authenticate using certificates for a particular URL, but still allow arbitrary clients to access the rest of the server?

To force clients to authenticate using certificates for a particular URL, you can use the per-directory reconfiguration features of [mod_ssl](#):

httpd.conf

```
SSLVerifyClient none
SSLCACertificateFile conf/ssl.crt/ca.crt
```

```
<Location /secure/area>
SSLVerifyClient require
SSLVerifyDepth 1
</Location>
```

How can I allow only clients who have certificates to access a particular URL, but allow all clients to access the rest of the server?

The key to doing this is checking that part of the client certificate matches what you expect. Usually this means checking all or part of the Distinguished Name (DN), to see if it contains some known string. There are two ways to do this, using either [mod_auth_basic](#) or [SSLRequire](#).

The [mod_auth_basic](#) method is generally required when the certificates are completely arbitrary, or when their DNs have no common fields (usually the organisation, etc.). In this case, you should establish a password database containing *all* clients allowed, as follows:

httpd.conf

```
SSLCACertificateFile conf/ssl.crt/ca.crt
SSLCACertificatePath conf/ssl.crt
SSLVerifyClient      none

<Directory /usr/local/apache2/htdocs/secure/area>
SSLVerifyClient      require
SSLVerifyDepth       5
SSLOptions            +FakeBasicAuth
SSLRequireSSL
AuthName              "Snake Oil Authentication"
AuthType              Basic
AuthBasicProvider     file
AuthUserFile          /usr/local/apache2/conf/httpd.passwd
Require               valid-user
</Directory>
```

The password used in this example is the DES encrypted string "password". See the [SSLOptions](#) docs for more information.

httpd.passwd

```
/C=DE/L=Munich/O=Snake Oil, Ltd./OU=Staff/CN=Foo:xxj31ZMTZzkVA  
/C=US/L=S.F./O=Snake Oil, Ltd./OU=CA/CN=Bar:xxj31ZMTZzkVA  
/C=US/L=L.A./O=Snake Oil, Ltd./OU=Dev/CN=Quux:xxj31ZMTZzkVA
```

When your clients are all part of a common hierarchy, which is encoded into the DN, you can match them more easily using [SSLRequire](#), as follows:

httpd.conf

```
SSLVerifyClient      none  
SSLCACertificateFile conf/ssl.crt/ca.crt  
SSLCACertificatePath conf/ssl.crt  
  
<Directory /usr/local/apache2/htdocs/secure/area>  
  SSLVerifyClient    require  
  SSLVerifyDepth     5  
  SSLOptions         +FakeBasicAuth  
  SSLRequireSSL  
  SSLRequire         %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \  
                    and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"}  
</Directory>
```

How can I require HTTPS with strong ciphers, and either basic authentication or client certificates, for access to part of the Intranet website, for clients coming from the Internet? I still want to allow plain HTTP access for clients on the Intranet.

These examples presume that clients on the Intranet have IPs in the range 192.168.1.0/24, and that the part of the Intranet website you want to allow internet access to is

/usr/local/apache2/htdocs/subarea. This configuration should remain outside of your HTTPS virtual host, so that it applies to both HTTPS and HTTP.

httpd.conf

```
SSLCACertificateFile conf/ssl.crt/company-ca.crt

<Directory /usr/local/apache2/htdocs>
# Outside the subarea only Intranet access is granted
Order          deny,allow
Deny           from all
Allow         from 192.168.1.0/24
</Directory>

<Directory /usr/local/apache2/htdocs/subarea>
# Inside the subarea any Intranet access is allowed
# but from the Internet only HTTPS + Strong-Cipher + Password
# or the alternative HTTPS + Strong-Cipher + Client-Certificate

# If HTTPS is used, make sure a strong cipher is used.
# Additionally allow client certs as alternative to basic auth.
SSLVerifyClient    optional
SSLVerifyDepth     1
SSLOptions         +FakeBasicAuth +StrictRequire
SSLRequire         %{SSL_CIPHER_USEKEYSIZE} >= 128

# Force clients from the Internet to use HTTPS
RewriteEngine      on
RewriteCond        %{REMOTE_ADDR} !^192\.168\.1\.[0-9]+$
RewriteCond        %{HTTPS} !=on
RewriteRule        .* - [F]

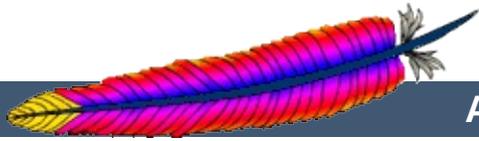
# Allow Network Access and/or Basic Auth
Satisfy            any

# Network Access Control
Order             deny,allow
Deny             from all
Allow           192.168.1.0/24

# HTTP Basic Authentication
AuthType         basic
AuthName         "Protected Intranet Area"
AuthBasicProvider file
AuthUserFile     conf/protected.passwd
Require         valid-user
</Directory>
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [SSL/TLS](#)

SSL/TLS Strong Encryption: FAQ

The wise man doesn't give the right answers, he poses the right questions.

-- Claude Levi-Strauss

This chapter is a collection of frequently asked questions (FAQ) and corresponding answers following the popular USENET tradition. Most of these questions occurred on the Newsgroup comp.infosystems.www.servers.unix or the mod_ssl Support Mailing List modssl-users@modssl.org. They are collected at this place to avoid answering the same questions over and over.

Please read this chapter at least once when installing mod_ssl or at least search for your problem here before submitting a problem report to the author.



- [What is the history of mod_ssl?](#)
- [mod_ssl and Wassenaar Arrangement?](#)

What is the history of mod_ssl?

The mod_ssl v1 package was initially created in April 1998 by [Ralf S. Engelschall](#) via porting [Ben Laurie's Apache-SSL 1.17](#) source patches for Apache 1.2.6 to Apache 1.3b6. Because of conflicts with Ben Laurie's development cycle it then was re-assembled from scratch for Apache 1.3.0 by merging the old mod_ssl 1.x with the newer Apache-SSL 1.18. From this point on mod_ssl lived its own life as mod_ssl v2. The first publicly released version was mod_ssl 2.0.0 from August 10th, 1998.

After US export restrictions on cryptographic software were loosened, [mod_ssl](#) became part of the Apache HTTP Server with the release of Apache httpd 2.

Is mod_ssl affected by the Wassenaar Arrangement?

First, let us explain what *Wassenaar* and its *Arrangement on Export Controls for Conventional Arms and Dual-Use Goods and Technologies* is: This is an international regime, established in 1995, to control trade in conventional arms and dual-use goods and technology. It replaced the previous *CoCom* regime. Further details on both the Arrangement and its signatories are available at <http://www.wassenaar.org/>.

In short, the aim of the Wassenaar Arrangement is to prevent the build up of military capabilities that threaten regional and international security and stability. The Wassenaar Arrangement controls the export of cryptography as a dual-use good, that is, something that has both military and civilian applications. However, the Wassenaar Arrangement also provides an exemption from export controls for mass-market software and free software.

In the current Wassenaar *List of Dual Use Goods and Technologies And Munitions*, under “GENERAL SOFTWARE NOTE (GSN)” it says “The Lists do not control “software” which is either: 1. [...] 2. “in the public domain”.” And under “DEFINITIONS OF TERMS USED IN THESE LISTS” we find “In the public domain” defined as ““technology” or “software” which has been made available without restrictions upon its further dissemination. Note: Copyright restrictions do not remove “technology” or “software” from being “in the public domain”.”

So, both mod_ssl and OpenSSL are “in the public domain” for the purposes of the Wassenaar Arrangement and its “*List of Dual Use Goods and Technologies And Munitions List*”, and thus not affected by its provisions.



- [Why do I get permission errors related to SSLMutex when I start Apache?](#)
- [Why does mod_ssl stop with the error "Failed to generate temporary 512 bit RSA private key" when I start Apache?](#)

Why do I get permission errors related to SSLMutex when I start Apache?

Errors such as ``mod_ssl: Child could not open SSLMutex lockfile /opt/apache/logs/ssl_mutex.18332 (System error follows) [...] System: Permission denied (errno: 13)`` are usually caused by overly restrictive permissions on the *parent* directories. Make sure that all parent directories (here `/opt`, `/opt/apache` and `/opt/apache/logs`) have the x-bit set for, at minimum, the UID under which Apache's children are running (see the [User](#) directive).

Why does mod_ssl stop with the error "Failed to generate temporary 512 bit RSA private key" when I start Apache?

Cryptographic software needs a source of unpredictable data to work correctly. Many open source operating systems provide a "randomness device" that serves this purpose (usually named `/dev/random`). On other systems, applications have to seed the OpenSSL Pseudo Random Number Generator (PRNG) manually with appropriate data before generating keys or performing public key encryption. As of version 0.9.5, the OpenSSL functions that need randomness report an error if the PRNG has not been seeded with at least 128 bits of randomness.

To prevent this error, `mod_ssl` has to provide enough entropy to the PRNG to allow it to work correctly. This can be done via the [SSLRandomSeed](#) directive.



- [Is it possible to provide HTTP and HTTPS from the same server?](#)
- [Which port does HTTPS use?](#)
- [How do I speak HTTPS manually for testing purposes?](#)
- [Why does the connection hang when I connect to my SSL-aware Apache server?](#)
- [Why do I get "Connection Refused" errors, when trying to access my newly installed Apache+mod_ssl server via HTTPS?](#)
- [Why are the SSL_XXX variables not available to my CGI & SSI scripts?](#)
- [How can I switch between HTTP and HTTPS in relative hyperlinks?](#)

Is it possible to provide HTTP and HTTPS from the same server?

Yes. HTTP and HTTPS use different server ports (HTTP binds to port 80, HTTPS to port 443), so there is no direct conflict between them. You can either run two separate server instances bound to these ports, or use Apache's elegant virtual hosting facility to create two virtual servers, both served by the same instance of Apache - one responding over HTTP to requests on port 80, and the other responding over HTTPS to requests on port 443.

Which port does HTTPS use?

You can run HTTPS on any port, but the standards specify port 443, which is where any HTTPS compliant browser will look by default. You can force your browser to look on a different port by specifying it in the URL. For example, if your server is set up to serve pages over HTTPS on port 8080, you can access them at `https://example.com:8080/`

How do I speak HTTPS manually for testing purposes?

While you usually just use

```
$ telnet localhost 80
GET / HTTP/1.0
```

for simple testing of Apache via HTTP, it's not so easy for HTTPS because of the SSL protocol between TCP and HTTP. With the help of OpenSSL's `s_client` command, however, you can do a similar check via HTTPS:

```
$ openssl s_client -connect localhost:443 -state -debug
GET / HTTP/1.0
```

Before the actual HTTP response you will receive detailed information about the SSL handshake. For a more general command line client which directly understands both HTTP and HTTPS, can perform GET and POST operations, can use a proxy, supports byte ranges, etc. you should have a look at the nifty [cURL](#) tool. Using this, you can check that Apache is responding correctly to requests via HTTP and HTTPS as follows:

```
$ curl http://localhost/
$ curl https://localhost/
```

Why does the connection hang when I connect to my SSL-aware Apache server?

This can happen when you try to connect to a HTTPS server (or virtual server) via HTTP (eg, using `http://example.com/` instead of `https://example.com`). It can also happen when trying to connect via HTTPS to a HTTP server (eg, using `https://example.com/` on a server which doesn't support HTTPS, or which supports it on a non-standard port). Make sure that you're connecting to a (virtual) server that supports SSL.

Why do I get "Connection Refused" messages, when trying to access my newly installed Apache+mod_ssl

server via HTTPS?

This error can be caused by an incorrect configuration. Please make sure that your [Listen](#) directives match your [<VirtualHost>](#) directives. If all else fails, please start afresh, using the default configuration provided by [mod_ssl](#).

Why are the SSL_XXX variables not available to my CGI & SSI scripts?

Please make sure you have ```SSLOptions +StdEnvVars``` enabled for the context of your CGI/SSI requests.

How can I switch between HTTP and HTTPS in relative hyperlinks?

Usually, to switch between HTTP and HTTPS, you have to use fully-qualified hyperlinks (because you have to change the URL scheme). Using [mod_rewrite](#) however, you can manipulate relative hyperlinks, to achieve the same effect.

```
RewriteEngine on
RewriteRule ^/(.*)_SSL$ https://%{SERVER_NAME}/$1 [R,L]
RewriteRule ^/(.*)_NOSSL$ http://%{SERVER_NAME}/$1 [R,L]
```

This rewrite ruleset lets you use hyperlinks of the form ``, to switch to HTTPS in a relative link. (Replace SSL with NOSSL to switch to HTTP.)



- [What are RSA Private Keys, CSRs and Certificates?](#)
- [Is there a difference on startup between a non-SSL-aware Apache and an SSL-aware Apache?](#)
- [How do I create a self-signed SSL Certificate for testing purposes?](#)
- [How do I create a real SSL Certificate?](#)
- [How do I create and use my own Certificate Authority \(CA\)?](#)
- [How can I change the pass-phrase on my private key file?](#)
- [How can I get rid of the pass-phrase dialog at Apache startup time?](#)
- [How do I verify that a private key matches its Certificate?](#)
- [Why do connections fail with an "alert bad certificate" error?](#)
- [Why does my 2048-bit private key not work?](#)
- [Why is client authentication broken after upgrading from SSLeay version 0.8 to 0.9?](#)
- [How can I convert a certificate from PEM to DER format?](#)
- [Why can't I find the `getca` or `getverisign` programs mentioned by Verisign, for installing my Verisign certificate?](#)
- [Can I use the Server Gated Cryptography \(SGC\) facility \(aka Verisign Global ID\) with `mod_ssl`?](#)
- [Why do browsers complain that they cannot verify my server certificate?](#)

What are RSA Private Keys, CSRs and Certificates?

An RSA private key file is a digital file that you can use to decrypt messages sent to you. It has a public component which you distribute (via your Certificate file) which allows people to encrypt those messages to you.

A Certificate Signing Request (CSR) is a digital file which contains your public key and your name. You send the CSR to a Certifying Authority (CA), who will convert it into a real Certificate, by signing it.

A Certificate contains your RSA public key, your name, the name of the CA, and is digitally signed by the CA. Browsers that know the CA can

verify the signature on that Certificate, thereby obtaining your RSA public key. That enables them to send messages which only you can decrypt.

See the [Introduction](#) chapter for a general description of the SSL protocol.

Is there a difference on startup between a non-SSL-aware Apache and an SSL-aware Apache?

Yes. In general, starting Apache with `mod_ssl` built-in is just like starting Apache without it. However, if you have a passphrase on your SSL private key file, a startup dialog will pop up which asks you to enter the pass phrase.

Having to manually enter the passphrase when starting the server can be problematic - for example, when starting the server from the system boot scripts. In this case, you can follow the steps [below](#) to remove the passphrase from your private key. Bear in mind that doing so brings additional security risks - proceed with caution!

How do I create a self-signed SSL Certificate for testing purposes?

1. Make sure OpenSSL is installed and in your PATH.
2. Run the following command, to create `server.key` and `server.crt` files:
\$ openssl req -new -x509 -nodes -out server.crt -keyout server.key

These can be used as follows in your `httpd.conf` file:

```
SSLCertificateFile    /path/to/this/se
SSLCertificateKeyFile /path/to/this/se
```

3. It is important that you are aware that this `server.key` does *not*

have any passphrase. To add a passphrase to the key, you should run the following command, and enter & verify the passphrase as requested.

```
$ openssl rsa -des3 -in server.key -out  
server.key.new  
$ mv server.key.new server.key
```

Please backup the server . key file, and the passphrase you entered, in a secure location.

How do I create a real SSL Certificate?

Here is a step-by-step description:

1. Make sure OpenSSL is installed and in your PATH.
2. Create a RSA private key for your Apache server (will be Triple-DES encrypted and PEM formatted):

```
$ openssl genrsa -des3 -out server.key 1024
```

Please backup this server . key file and the pass-phrase you entered in a secure location. You can see the details of this RSA private key by using the command:

```
$ openssl rsa -noout -text -in server.key
```

If necessary, you can also create a decrypted PEM version (not recommended) of this RSA private key with:

```
$ openssl rsa -in server.key -out  
server.key.unsecure
```

3. Create a Certificate Signing Request (CSR) with the server RSA

private key (output will be PEM formatted):

```
$ openssl req -new -key server.key -out server.csr
```

Make sure you enter the FQDN ("Fully Qualified Domain Name") of the server when OpenSSL prompts you for the "CommonName", i.e. when you generate a CSR for a website which will be later accessed via `https://www.foo.dom/`, enter "www.foo.dom" here. You can see the details of this CSR by using

```
$ openssl req -noout -text -in server.csr
```

4. You now have to send this Certificate Signing Request (CSR) to a Certifying Authority (CA) to be signed. Once the CSR has been signed, you will have a real Certificate, which can be used by Apache. You can have a CSR signed by a commercial CA, or you can create your own CA to sign it.
Commercial CAs usually ask you to post the CSR into a web form, pay for the signing, and then send a signed Certificate, which you can store in a server.crt file. For more information about commercial CAs see the following locations:

1. Verisign
<http://digitalid.verisign.com/server/apacheNotice.htm>
2. Thawte
<http://www.thawte.com/>
3. CertiSign Certificadora Digital Ltda.
<http://www.certisign.com.br>
4. IKS GmbH
<http://www.iks-jena.de/leistungen/ca/>
5. Uptime Commerce Ltd.
<http://www.uptimecommerce.com>

6. BelSign NV/SA

<http://www.belsign.be>

For details on how to create your own CA, and use this to sign a CSR, see [below](#).

Once your CSR has been signed, you can see the details of the Certificate as follows:

```
$ openssl x509 -noout -text -in server.crt
```

5. You should now have two files: `server.key` and `server.crt`. These can be used as follows in your `httpd.conf` file:

```
SSLCertificateFile    /path/to/this/server.c
SSLCertificateKeyFile /path/to/this/server.k
```

The `server.csr` file is no longer needed.

How do I create and use my own Certificate Authority (CA)?

The short answer is to use the `CA.sh` or `CA.pl` script provided by OpenSSL. Unless you have a good reason not to, you should use these for preference. If you cannot, you can create a self-signed Certificate as follows:

1. Create a RSA private key for your server (will be Triple-DES encrypted and PEM formatted):

```
$ openssl genrsa -des3 -out server.key 1024
```

Please backup this `server.key` file and the pass-phrase you entered in a secure location. You can see the details of this RSA private key by using the command:

```
$ openssl rsa -noout -text -in server.key
```

If necessary, you can also create a decrypted PEM version (not recommended) of this RSA private key with:

```
$ openssl rsa -in server.key -out  
server.key.unsecure
```

2. Create a self-signed Certificate (X509 structure) with the RSA key you just created (output will be PEM formatted):

```
$ openssl req -new -x509 -nodes -sha1 -days 365 -  
key server.key -out server.crt
```

This signs the server CSR and results in a `server.crt` file. You can see the details of this Certificate using:

```
$ openssl x509 -noout -text -in server.crt
```

How can I change the pass-phrase on my private key file?

You simply have to read it with the old pass-phrase and write it again, specifying the new pass-phrase. You can accomplish this with the following commands:

```
$ openssl rsa -des3 -in server.key -out server.key.new  
$ mv server.key.new server.key
```

The first time you're asked for a PEM pass-phrase, you should enter the old pass-phrase. After that, you'll be asked again to enter a pass-phrase - this time, use the new pass-phrase. If you are asked to verify the pass-phrase, you'll need to enter the new pass-phrase a second time.

How can I get rid of the pass-phrase dialog at Apache

startup time?

The reason this dialog pops up at startup and every re-start is that the RSA private key inside your server.key file is stored in encrypted format for security reasons. The pass-phrase is needed to decrypt this file, so it can be read and parsed. Removing the pass-phrase removes a layer of security from your server - proceed with caution!

1. Remove the encryption from the RSA private key (while keeping a backup copy of the original file):

```
$ cp server.key server.key.org  
$ openssl rsa -in server.key.org -out server.key
```

2. Make sure the server.key file is only readable by root:

```
$ chmod 400 server.key
```

Now server . key contains an unencrypted copy of the key. If you point your server at this file, it will not prompt you for a pass-phrase. HOWEVER, if anyone gets this key they will be able to impersonate you on the net. PLEASE make sure that the permissions on this file are such that only root or the web server user can read it (preferably get your web server to start as root but run as another user, and have the key readable only by root).

As an alternative approach you can use the ``SSLPassPhraseDialog exec:/path/to/program" facility. Bear in mind that this is neither more nor less secure, of course.

How do I verify that a private key matches its Certificate?

A private key contains a series of numbers. Two of these numbers form the "public key", the others are part of the "private key". The "public key"

bits are included when you generate a CSR, and subsequently form part of the associated Certificate.

To check that the public key in your Certificate matches the public portion of your private key, you simply need to compare these numbers. To view the Certificate and the key run the commands:

```
$ openssl x509 -noout -text -in server.crt  
$ openssl rsa -noout -text -in server.key
```

The `modulus' and the `public exponent' portions in the key and the Certificate must match. As the public exponent is usually 65537 and it's difficult to visually check that the long modulus numbers are the same, you can use the following approach:

```
$ openssl x509 -noout -modulus -in server.crt |  
openssl md5  
$ openssl rsa -noout -modulus -in server.key | openssl  
md5
```

This leaves you with two rather shorter numbers to compare. It is, in theory, possible that these numbers may be the same, without the modulus numbers being the same, but the chances of this are overwhelmingly remote.

Should you wish to check to which key or certificate a particular CSR belongs you can perform the same calculation on the CSR as follows:

```
$ openssl req -noout -modulus -in server.csr | openssl  
md5
```

Why do connections fail with an "alert bad certificate" error?

Errors such as OpenSSL: error:14094412: SSL routines:SSL3_READ_BYTES:sslv3 alert bad certificate in

the SSL logfile, are usually caused by a browser which is unable to handle the server certificate/private-key. For example, Netscape Navigator 3.x is unable to handle RSA key lengths not equal to 1024 bits.

Why does my 2048-bit private key not work?

The private key sizes for SSL must be either 512 or 1024 bits, for compatibility with certain web browsers. A keysize of 1024 bits is recommended because keys larger than 1024 bits are incompatible with some versions of Netscape Navigator and Microsoft Internet Explorer, and with other browsers that use RSA's BSAFE cryptography toolkit.

Why is client authentication broken after upgrading from SSLeay version 0.8 to 0.9?

The CA certificates under the path you configured with `SSLCACertificatePath` are found by SSLeay through hash symlinks. These hash values are generated by the ``openssl x509 -noout -hash'` command. However, the algorithm used to calculate the hash for a certificate changed between SSLeay 0.8 and 0.9. You will need to remove all old hash symlinks and create new ones after upgrading. Use the Makefile provided by [mod_ssl](#).

How can I convert a certificate from PEM to DER format?

The default certificate format for SSLeay/OpenSSL is PEM, which is simply Base64 encoded DER, with header and footer lines. For some applications (e.g. Microsoft Internet Explorer) you need the certificate in plain DER format. You can convert a PEM file `cert.pem` into the corresponding DER file `cert.der` using the following command: `$ openssl x509 -in cert.pem -out cert.der -outform DER`

Why can't I find the `getca` or `getverisign` programs mentioned by Verisign, for installing my Verisign certificate?

Verisign has never provided specific instructions for Apache+mod_ssl. The instructions provided are for C2Net's Stronghold (a commercial Apache based server with SSL support).

To install your certificate, all you need to do is to save the certificate to a file, and give the name of that file to the [SSLCertificateFile](#) directive. You will also need to give it the key file. For more information, see the [SSLCertificateKeyFile](#) directive.

Can I use the Server Gated Cryptography (SGC) facility (aka Verisign Global ID) with mod_ssl?

Yes. [mod_ssl](#) has included support for the SGC facility since version 2.1. No special configuration is required - just use the Global ID as your server certificate. The *step up* of the clients is then automatically handled by [mod_ssl](#) at run-time.

Why do browsers complain that they cannot verify my server certificate?

One reason this might happen is because your server certificate is signed by an intermediate CA. Various CAs, such as Verisign or Thawte, have started signing certificates not with their root certificate but with intermediate certificates.

Intermediate CA certificates lie between the root CA certificate (which is installed in the browsers) and the server certificate (which you installed on the server). In order for the browser to be able to traverse and verify the trust chain from the server certificate to the root certificate it needs need to be given the intermediate certificates. The CAs should be able to provide you such intermediate certificate packages that can be installed on the server.

You need to include those intermediate certificates with the [SSLCertificateChainFile](#) directive.



- [Why do I get lots of random SSL protocol errors under heavy server load?](#)
- [Why does my webserver have a higher load, now that it serves SSL encrypted traffic?](#)
- [Why do HTTPS connections to my server sometimes take up to 30 seconds to establish a connection?](#)
- [What SSL Ciphers are supported by mod_ssl?](#)
- [Why do I get ``no shared cipher" errors, when trying to use Anonymous Diffie-Hellman \(ADH\) ciphers?](#)
- [Why do I get a 'no shared ciphers' error when connecting to my newly installed server?](#)
- [Why can't I use SSL with name-based/non-IP-based virtual hosts?](#)
- [Is it possible to use Name-Based Virtual Hosting to identify different SSL virtual hosts?](#)
- [How do I get SSL compression working?](#)
- [When I use Basic Authentication over HTTPS the lock icon in Netscape browsers stays unlocked when the dialog pops up. Does this mean the username/password is being sent unencrypted?](#)
- [Why do I get I/O errors when connecting via HTTPS to an Apache+mod_ssl server with Microsoft Internet Explorer \(MSIE\)?](#)
- [Why do I get I/O errors, or the message "Netscape has encountered bad data from the server", when connecting via HTTPS to an Apache+mod_ssl server with Netscape Navigator?](#)
- [Why do I get handshake failures with Java-based clients when using a certificate with more than 1024 bits?](#)

Why do I get lots of random SSL protocol errors under heavy server load?

There can be a number of reasons for this, but the main one is problems with the SSL session Cache specified by the [SSLSessionCache](#) directive. The DBM session cache is the most likely source of the problem, so using the SHM session cache (or no cache at all) may help.

Why does my webserver have a higher load, now that it serves SSL encrypted traffic?

SSL uses strong cryptographic encryption, which necessitates a lot of number crunching. When you request a webpage via HTTPS, everything (even the images) is encrypted before it is transferred. So increased HTTPS traffic leads to load increases.

Why do HTTPS connections to my server sometimes take up to 30 seconds to establish a connection?

This is usually caused by a `/dev/random` device for [SSLRandomSeed](#) which blocks the `read(2)` call until enough entropy is available to service the request. More information is available in the reference manual for the [SSLRandomSeed](#) directive.

What SSL Ciphers are supported by mod_ssl?

Usually, any SSL ciphers supported by the version of OpenSSL in use, are also supported by [mod_ssl](#). Which ciphers are available can depend on the way you built OpenSSL. Typically, at least the following ciphers are supported:

1. RC4 with MD5
2. RC4 with MD5 (export version restricted to 40-bit key)
3. RC2 with MD5
4. RC2 with MD5 (export version restricted to 40-bit key)
5. IDEA with MD5
6. DES with MD5
7. Triple-DES with MD5

To determine the actual list of ciphers available, you should run the following:

```
$ openssl ciphers -v
```

Why do I get "no shared cipher" errors, when trying to use Anonymous Diffie-Hellman (ADH) ciphers?

By default, OpenSSL does *not* allow ADH ciphers, for security reasons. Please be sure you are aware of the potential side-effects if you choose to enable these ciphers.

In order to use Anonymous Diffie-Hellman (ADH) ciphers, you must build OpenSSL with ```-DSSL_ALLOW_ADH"`, and then add ```ADH"` into your [SSLCipherSuite](#).

Why do I get a 'no shared ciphers' error when connecting to my newly installed server?

Either you have made a mistake with your [SSLCipherSuite](#) directive (compare it with the pre-configured example in `httpd.conf-dist`) or you chose to use DSA/DH algorithms instead of RSA when you generated your private key and ignored or overlooked the warnings. If you have chosen DSA/DH, then your server cannot communicate using RSA-based SSL ciphers (at least until you configure an additional RSA-based certificate/key pair). Modern browsers like NS or IE can only communicate over SSL using RSA ciphers. The result is the "no shared ciphers" error. To fix this, regenerate your server certificate/key pair, using the RSA algorithm.

Why can't I use SSL with name-based/non-IP-based virtual hosts?

The reason is very technical, and a somewhat "chicken and egg" problem. The SSL protocol layer stays below the HTTP protocol layer and encapsulates HTTP. When an SSL connection (HTTPS) is established Apache/mod_ssl has to negotiate the SSL protocol parameters with the client. For this, mod_ssl has to consult the

configuration of the virtual server (for instance it has to look for the cipher suite, the server certificate, etc.). But in order to go to the correct virtual server Apache has to know the Host HTTP header field. To do this, the HTTP request header has to be read. This cannot be done before the SSL handshake is finished, but the information is needed in order to complete the SSL handshake phase. See the next question for how to circumvent this issue.

Note that if you have a wildcard SSL certificate, or a certificate that has multiple hostnames on it using subjectAltName fields, you can use SSL on name-based virtual hosts without further workarounds.

Why is it not possible to use Name-Based Virtual Hosting to identify different SSL virtual hosts?

Name-Based Virtual Hosting is a very popular method of identifying different virtual hosts. It allows you to use the same IP address and the same port number for many different sites. When people move on to SSL, it seems natural to assume that the same method can be used to have lots of different SSL virtual hosts on the same server.

It is possible, but only if using a 2.2.12 or later web server, built with 0.9.8j or later OpenSSL. This is because it requires a feature that only the most recent revisions of the SSL specification added, called Server Name Indication (SNI).

Note that if you have a wildcard SSL certificate, or a certificate that has multiple hostnames on it using subjectAltName fields, you can use SSL on name-based virtual hosts without further workarounds.

The reason is that the SSL protocol is a separate layer which encapsulates the HTTP protocol. So the SSL session is a separate transaction, that takes place before the HTTP session has begun. The server receives an SSL request on IP address X and port Y (usually 443). Since the SSL request did not contain any Host: field, the server had no

way to decide which SSL virtual host to use. Usually, it just used the first one it found which matched the port and IP address specified.

If you are using a version of the web server and OpenSSL that support SNI, though, and the client's browser also supports SNI, then the hostname is included in the original SSL request, and the web server can select the correct SSL virtual host.

You can, of course, use Name-Based Virtual Hosting to identify many non-SSL virtual hosts (all on port 80, for example) and then have a single SSL virtual host (on port 443). But if you do this, you must make sure to put the non-SSL port number on the NameVirtualHost directive, e.g.

```
NameVirtualHost 192.168.1.1:80
```

Other workaround solutions include:

Using separate IP addresses for different SSL hosts. Using different port numbers for different SSL hosts.

How do I get SSL compression working?

Although SSL compression negotiation was defined in the specification of SSLv2 and TLS, it took until May 2004 for RFC 3749 to define DEFLATE as a negotiable standard compression method.

OpenSSL 0.9.8 started to support this by default when compiled with the `zlib` option. If both the client and the server support compression, it will be used. However, most clients still try to initially connect with an SSLv2 Hello. As SSLv2 did not include an array of preferred compression algorithms in its handshake, compression cannot be negotiated with these clients. If the client disables support for SSLv2, either an SSLv3 or TLS Hello may be sent, depending on which SSL library is used, and compression may be set up. You can verify whether clients make use of SSL compression by logging the `%{SSL_COMPRESS_METHOD}x` variable.

When I use Basic Authentication over HTTPS the lock icon in Netscape browsers stays unlocked when the dialog pops up. Does this mean the username/password is being sent unencrypted?

No, the username/password is transmitted encrypted. The icon in Netscape browsers is not actually synchronized with the SSL/TLS layer. It only toggles to the locked state when the first part of the actual webpage data is transferred, which may confuse people. The Basic Authentication facility is part of the HTTP layer, which is above the SSL/TLS layer in HTTPS. Before any HTTP data communication takes place in HTTPS, the SSL/TLS layer has already completed its handshake phase, and switched to encrypted communication. So don't be confused by this icon.

Why do I get I/O errors when connecting via HTTPS to an Apache+mod_ssl server with Microsoft Internet Explorer (MSIE)?

The first reason is that the SSL implementation in some MSIE versions has some subtle bugs related to the HTTP keep-alive facility and the SSL close notify alerts on socket connection close. Additionally the interaction between SSL and HTTP/1.1 features are problematic in some MSIE versions. You can work around these problems by forcing Apache not to use HTTP/1.1, keep-alive connections or send the SSL close notify messages to MSIE clients. This can be done by using the following directive in your SSL-aware virtual host section:

```
SetEnvIf User-Agent ".*MSIE.*" \
nokeepalive ssl-unclean-shutdown \
downgrade-1.0 force-response-1.0
```

Further, some MSIE versions have problems with particular ciphers. Unfortunately, it is not possible to implement a MSIE-specific workaround for this, because the ciphers are needed as early as the SSL handshake phase. So a MSIE-specific [SetEnvIf](#) won't solve these problems.

Instead, you will have to make more drastic adjustments to the global parameters. Before you decide to do this, make sure your clients really have problems. If not, do not make these changes - they will affect *all* your clients, MSIE or otherwise.

The next problem is that 56bit export versions of MSIE 5.x browsers have a broken SSLv3 implementation, which interacts badly with OpenSSL versions greater than 0.9.4. You can accept this and require your clients to upgrade their browsers, you can downgrade to OpenSSL 0.9.4 (not advised), or you can work around this, accepting that your workaround will affect other browsers too:

```
SSLProtocol all -SSLv3
```

will completely disables the SSLv3 protocol and allow those browsers to work. A better workaround is to disable only those ciphers which cause trouble.

```
SSLCipherSuite  
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP
```

This also allows the broken MSIE versions to work, but only removes the newer 56bit TLS ciphers.

Another problem with MSIE 5.x clients is that they refuse to connect to URLs of the form `ht tps://12.34.56.78/` (where IP-addresses are used instead of the hostname), if the server is using the Server Gated Cryptography (SGC) facility. This can only be avoided by using the fully qualified domain name (FQDN) of the website in hyperlinks instead, because MSIE 5.x has an error in the way it handles the SGC negotiation.

And finally there are versions of MSIE which seem to require that an SSL session can be reused (a totally non standard-conforming behaviour, of course). Connecting with those MSIE versions only work if a SSL session

cache is used. So, as a work-around, make sure you are using a session cache (see the [SSLSessionCache](#) directive).

Why do I get I/O errors, or the message "Netscape has encountered bad data from the server", when connecting via HTTPS to an Apache+mod_ssl server with Netscape Navigator?

This usually occurs when you have created a new server certificate for a given domain, but had previously told your browser to always accept the old server certificate. Once you clear the entry for the old certificate from your browser, everything should be fine. Netscape's SSL implementation is correct, so when you encounter I/O errors with Netscape Navigator it is usually caused by the configured certificates.

Why do I get handshake failures with Java-based clients when using a certificate with more than 1024 bits?

Beginning with version 2.2.30, [mod_ssl](#) will use DH parameters which include primes with lengths of more than 1024 bits. Java 7 and earlier limit their support for DH prime sizes to a maximum of 1024 bits, however.

If your Java-based client aborts with exceptions such as `java.lang.RuntimeException: Could not generate DH keypair` and `java.security.InvalidAlgorithmParameterException: Prime size must be multiple of 64, and can only range from 512 to 1024 (inclusive)`, and `httpd logs tlsv1 alert internal error (SSL alert number 80)` (at [LogLevel](#) info or higher), you can either rearrange `mod_ssl`'s cipher list with [SSLCipherSuite](#) (possibly in conjunction with [SSLHonorCipherOrder](#)), or you can use custom DH parameters with a 1024-bit prime, which will always have precedence over any of the built-in DH parameters.

To generate custom DH parameters, use the `openssl dhparam 1024` command. Alternatively, you can use the following standard 1024-bit DH parameters from [RFC 2409](#), section 6.2:

```
-----BEGIN DH PARAMETERS-----
MIGHAoGBAP/////////yQ/aoiFowjTExmKLGnwc0SkCTgiKZ8x0Agu+pjsTmyJR
Sgh5jjQE3e+VGbPN0kMbMCskbfJfFDdP4TVtbVHCreSFtXZiXn7G9ExC6aY37WsL
/1y29Aa37e44a/taiZ+lrp8kEXxLH+ZJKGZR70ZTgf/////////AgEC
-----END DH PARAMETERS-----
```

Add the custom parameters including the "BEGIN DH PARAMETERS" and "END DH PARAMETERS" lines to the end of the first certificate file you have configured using the [SSLCertificateFile](#) directive.



- [What information resources are available in case of mod_ssl problems?](#)
- [What support contacts are available in case of mod_ssl problems?](#)
- [What information should I provide when writing a bug report?](#)
- [I had a core dump, can you help me?](#)
- [How do I get a backtrace, to help find the reason for my core dump?](#)

What information resources are available in case of mod_ssl problems?

The following information resources are available. In case of problems you should search here first.

Answers in the User Manual's F.A.Q. List ([this](#))

http://httpd.apache.org/docs/2.2/ssl/ssl_faq.html

First check the F.A.Q. ([this text](#)). If your problem is a common one, it may have been answered several times before, and been included in this doc.

Postings from the modssl-users Support Mailing List

<http://www.modssl.org/support/>

Search for your problem in the archives of the modssl-users mailing list. You're probably not the first person to have had this problem!

What support contacts are available in case of mod_ssl problems?

The following lists all support possibilities for mod_ssl, in order of preference. Please go through these possibilities *in this order* - don't just pick the one you like the look of.

1. *Send a Problem Report to the modssl-users Support Mailing List*
modssl-users@modssl.org

This is the preferred way of submitting your problem report, because this way, others can see the problem, and learn from any answers.

You must subscribe to the list first, but you can then easily discuss your problem with both the author and the whole mod_ssl user community.

2. *Send a Problem Report to the Apache httpd Users Support Mailing List*

users@httpd.apache.org

This is the second way of submitting your problem report. Again, you must subscribe to the list first, but you can then easily discuss your problem with the whole Apache httpd user community.

3. *Write a Problem Report in the Bug Database*

http://httpd.apache.org/bug_report.html

This is the last way of submitting your problem report. You should only do this if you've already posted to the mailing lists, and had no success. Please follow the instructions on the above page *carefully*.

What information should I provide when writing a bug report?

You should always provide at least the following information:

Apache and OpenSSL version information

The Apache version can be determined by running `httpd -v`. The OpenSSL version can be determined by running `openssl version`. Alternatively, if you have Lynx installed, you can run the command `lynx -mime_header http://localhost/ | grep Server` to gather this information in a single step.

The details on how you built and installed Apache+mod_ssl+OpenSSL

For this you can provide a logfile of your terminal session which shows the configuration and install steps. If this is not possible, you should at least provide the [configure](#) command line you used.

In case of core dumps please include a Backtrace

If your Apache+mod_ssl+OpenSSL dumps its core, please attach a

stack-frame `backtrace`" (see [below](#) for information on how to get this). This information is required in order to find a reason for your core dump.

A detailed description of your problem

Don't laugh, we really mean it! Many problem reports don't include a description of what the actual problem is. Without this, it's very difficult for anyone to help you. So, it's in your own interest (you want the problem be solved, don't you?) to include as much detail as possible, please. Of course, you should still include all the essentials above too.

I had a core dump, can you help me?

In general no, at least not unless you provide more details about the code location where Apache dumped core. What is usually always required in order to help you is a backtrace (see next question). Without this information it is mostly impossible to find the problem and help you in fixing it.

How do I get a backtrace, to help find the reason for my core dump?

Following are the steps you will need to complete, to get a backtrace:

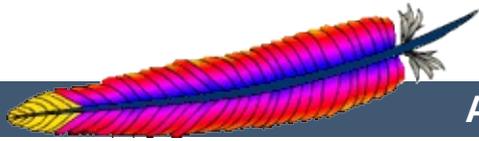
1. Make sure you have debugging symbols available, at least in Apache. On platforms where you use GCC/GDB, you will have to build Apache+mod_ssl with `OPTIM="-g -ggdb3"` to get this. On other platforms at least `OPTIM="-g"` is needed.
2. Start the server and try to reproduce the core-dump. For this you may want to use a directive like `CoreDumpDirectory /tmp` to make sure that the core-dump file can be written. This should result in a `/tmp/core` or `/tmp/httpd.core` file. If you don't get one of these, try running your server under a non-root UID. Many modern kernels do not allow a process to dump core after it has done a

`setuid()` (unless it does an `exec()`) for security reasons (there can be privileged information left over in memory). If necessary, you can run `/path/to/httpd -X` manually to force Apache to not fork.

3. Analyze the core-dump. For this, run `gdb /path/to/httpd /tmp/httpd.core` or a similar command. In GDB, all you have to do then is to enter `bt`, and voila, you get the backtrace. For other debuggers consult your local debugger manual.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [How-To / Tutorials](#)

Authentication and Authorization

Authentication is any process by which you verify that someone is who they claim they are. Authorization is any process by which someone is allowed to be where they want to go, or to have information that they want to have.

For general access control, see the [Access Control How-To](#).



Authentication and Authorization

There are three types of modules involved in the authentication and authorization process. You will usually need to choose at least one module from each group.

- Authentication type (see the [AuthType](#) directive)
 - [mod_auth_basic](#)
 - [mod_auth_digest](#)
- Authentication provider (see the [AuthBasicProvider](#) and [AuthDigestProvider](#) directives)
 - [mod_authn_anon](#)
 - [mod_authn_dbd](#)
 - [mod_authn_dbm](#)
 - [mod_authn_file](#)
 - [mod_authnz_ldap](#)
- Authorization (see the [Require](#) directive)
 - [mod_authnz_ldap](#)
 - [mod_authz_dbm](#)
 - [mod_authz_groupfile](#)
 - [mod_authz_host](#)
 - [mod_authz_owner](#)
 - [mod_authz_user](#)

The module [mod_authnz_ldap](#) is both an authentication and authorization provider. The module [mod_authn_alias](#) is not an authentication provider in itself, but allows other authentication providers to be configured in a flexible manner.

The module [mod_authz_host](#) provides authorization and access control based on hostname, IP address or characteristics of the request, but is not part of the authentication provider system.

You probably also want to take a look at the [Access Control](#) howto, which discusses the various ways to control access to your server.



If you have information on your web site that is sensitive or intended for only a small group of people, the techniques in this article will help you make sure that the people that see those pages are the people that you wanted to see them.

This article covers the "standard" way of protecting parts of your web site that most of you are going to use.

Note:

If your data really needs to be secure, consider using [mod_ssl](#) in addition to any authentication.



The Prerequisites

The directives discussed in this article will need to go either in your main server configuration file (typically in a `<Directory>` section), or in per-directory configuration files (`.htaccess` files).

If you plan to use `.htaccess` files, you will need to have a server configuration that permits putting authentication directives in these files. This is done with the `AllowOverride` directive, which specifies which directives, if any, may be put in per-directory configuration files.

Since we're talking here about authentication, you will need an `AllowOverride` directive like the following:

```
AllowOverride AuthConfig
```

Or, if you are just going to put the directives directly in your main server configuration file, you will of course need to have write permission to that file.

And you'll need to know a little bit about the directory structure of your server, in order to know where some files are kept. This should not be terribly difficult, and I'll try to make this clear when we come to that point.



Here's the basics of password protecting a directory on your server.

First, you need to create a password file. Exactly how you do this will vary depending on what authentication provider you have chosen. More on that later. To start with, we'll use a text password file.

This file should be placed somewhere not accessible from the web. This is so that folks cannot download the password file. For example, if your documents are served out of `/usr/local/apache/htdocs` you might want to put the password file(s) in `/usr/local/apache/passwd`.

To create the file, use the `htpasswd` utility that came with Apache. This will be located in the `bin` directory of wherever you installed Apache. If you have installed Apache from a third-party package, it may be in your execution path.

To create the file, type:

```
htpasswd -c /usr/local/apache/passwd/passwords rbowen
```

`htpasswd` will ask you for the password, and then ask you to type it again to confirm it:

```
# htpasswd -c /usr/local/apache/passwd/passwords rbowen
New password: mypassword
Re-type new password: mypassword
Adding password for user rbowen
```

If `htpasswd` is not in your path, of course you'll have to type the full path to the file to get it to run. With a default installation, it's located at `/usr/local/apache2/bin/htpasswd`

Next, you'll need to configure the server to request a password and tell the server which users are allowed access. You can do this either by editing the `httpd.conf` file or using an `.htaccess` file. For example, if

you wish to protect the directory `/usr/local/apache/htdocs/secret`, you can use the following directives, either placed in the file `/usr/local/apache/htdocs/secret/.htaccess`, or placed in `httpd.conf` inside a `<Directory /usr/local/apache/apache/htdocs/secret>` section.

```
AuthType Basic
AuthName "Restricted Files"
# (Following line optional)
AuthBasicProvider file
AuthUserFile /usr/local/apache/passwd/passwords
Require user rbowen
```

Let's examine each of those directives individually. The [AuthType](#) directive selects that method that is used to authenticate the user. The most common method is `Basic`, and this is the method implemented by [mod_auth_basic](#). It is important to be aware, however, that `Basic` authentication sends the password from the client to the server unencrypted. This method should therefore not be used for highly sensitive data, unless accompanied by [mod_ssl](#). Apache supports one other authentication method: `AuthType Digest`. This method is implemented by [mod_auth_digest](#) and was intended to be more secure. This is no longer the case and the connection should be encrypted with [mod_ssl](#) instead.

The [AuthName](#) directive sets the *Realm* to be used in the authentication. The realm serves two major functions. First, the client often presents this information to the user as part of the password dialog box. Second, it is used by the client to determine what password to send for a given authenticated area.

So, for example, once a client has authenticated in the "Restricted Files" area, it will automatically retry the same password for any area on the same server that is marked with the "Restricted Files"

Realm. Therefore, you can prevent a user from being prompted more than once for a password by letting multiple restricted areas share the same realm. Of course, for security reasons, the client will always need to ask again for the password whenever the hostname of the server changes.

The [AuthBasicProvider](#) is, in this case, optional, since `file` is the default value for this directive. You'll need to use this directive if you are choosing a different source for authentication, such as [mod_authn_dbm](#) or [mod_authn_dbd](#).

The [AuthUserFile](#) directive sets the path to the password file that we just created with [htpasswd](#). If you have a large number of users, it can be quite slow to search through a plain text file to authenticate the user on each request. Apache also has the ability to store user information in fast database files. The [mod_authn_dbm](#) module provides the [AuthDBMUserFile](#) directive. These files can be created and manipulated with the [dbmmanage](#) program. Many other types of authentication options are available from third party modules in the [Apache Modules Database](#).

Finally, the [Require](#) directive provides the authorization part of the process by setting the user that is allowed to access this region of the server. In the next section, we discuss various ways to use the [Require](#) directive.



The **Satisfy** directive can be used to specify that several criteria may be considered when trying to decide if a particular user will be granted admission. Satisfy can take as an argument one of two options - `all` or `any`. By default, it is assumed that the value is `all`. This means that if several criteria are specified, then all of them must be met in order for someone to get in. However, if set to `any`, then several criteria may be specified, but if the user satisfies any of these, then they will be granted entrance.

An example of this is using access control to assure that, although a resource is password protected from outside your network, all hosts inside the network will be given unauthenticated access to the resource. This would be accomplished by using the Satisfy directive, as shown below.

```
<Directory /usr/local/apache/htdocs/sekrit>
  AuthType Basic
  AuthName intranet
  AuthUserFile /www/passwd/users
  AuthGroupFile /www/passwd/groups
  Require group customers
  Order allow,deny
  Allow from internal.com
  Satisfy any
</Directory>
```



The directives above only let one person (specifically someone with a username of rbowen) into the directory. In most cases, you'll want to let more than one person in. This is where the [AuthGroupFile](#) comes in.

If you want to let more than one person in, you'll need to create a group file that associates group names with a list of users in that group. The format of this file is pretty simple, and you can create it with your favorite editor. The contents of the file will look like this:

```
GroupName: rbowen dpitts sungo rshersey
```

That's just a list of the members of the group in a long line separated by spaces.

To add a user to your already existing password file, type:

```
htpasswd /usr/local/apache/passwd/passwords dpitts
```

You'll get the same response as before, but it will be appended to the existing file, rather than creating a new file. (It's the `-c` that makes it create a new password file).

Now, you need to modify your `.htaccess` file or [<Directory>](#) block to look like the following:

```
AuthType Basic
AuthName "By Invitation Only"
# Optional line:
AuthBasicProvider file
AuthUserFile /usr/local/apache/passwd/passwords
AuthGroupFile /usr/local/apache/passwd/groups
Require group GroupName
```

Now, anyone that is listed in the group `GroupName`, and has an entry in the password file, will be let in, if they type the correct password.

There's another way to let multiple users in that is less specific. Rather than creating a group file, you can just use the following directive:

```
Require valid-user
```

Using that rather than the `Require user` r Bowen line will allow anyone in that is listed in the password file, and who correctly enters their password.



Performance Problems

Because of the way that Basic authentication is specified, your username and password must be verified every time you request a document from the server. This is even if you're reloading the same page, and for every image on the page (if they come from a protected directory). As you can imagine, this slows things down a little. The amount that it slows things down is proportional to the size of the password file, because it has to open up that file, and go down the list of users until it gets to your name. And it has to do this every time a page is loaded.

A consequence of this is that there's a practical limit to how many users you can put in one password file. This limit will vary depending on the performance of your particular server machine, but you can expect to see slowdowns once you get above a few hundred entries, and may wish to consider a different authentication method at that time.



Alternative password storage

Because storing passwords in plain text files has the above problems, you may wish to store your passwords somewhere else, such as in a database.

[mod_authn_dbm](#) and [mod_authn_dbd](#) are two modules which make this possible. Rather than selecting [AuthBasicProvider](#) file, instead you can choose dbm or dbd as your storage format.

To select a dbd file rather than a text file, for example:

```
<Directory /www/docs/private>
  AuthName "Private"
  AuthType Basic
  AuthBasicProvider dbm
  AuthDBMUserFile /www/passwords/passwd.dbm
  Require valid-user
</Directory>
```

Other options are available. Consult the [mod_authn_dbm](#) documentation for more details.



MORE INFORMATION

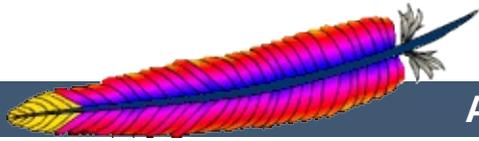
You should also read the documentation for [mod_auth_basic](#) and [mod_authz_host](#) which contain some more information about how this all works. [mod_authn_alias](#) can also help in simplifying certain authentication configurations.

The various ciphers supported by Apache for authentication data are explained in [Password Encryptions](#).

And you may want to look at the [Access Control](#) howto, which discusses a number of related topics.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [How-To / Tutorials](#)

Access Control

Access control refers to any means of controlling access to any resource. This is separate from [authentication and authorization](#).



Request Handling and Directives

Access control can be done by several different modules. The most important of these is [mod_authz_host](#). Other modules discussed in this document include [mod_setenvif](#) and [mod_rewrite](#).



Restricting Access by Host

If you wish to restrict access to portions of your site based on the host address of your visitors, this is most easily done using `mod_authz_host`.

The `Allow` and `Deny` directives let you allow and deny access based on the host name, or host address, of the machine requesting a document. The `Order` directive goes hand-in-hand with these two, and tells Apache in which order to apply the filters.

The usage of these directives is:

```
Allow from address
```

where *address* is an IP address (or a partial IP address) or a fully qualified domain name (or a partial domain name); you may provide multiple addresses or domain names, if desired.

For example, if you have someone spamming your message board, and you want to keep them out, you could do the following:

```
Deny from 10.252.46.165
```

Visitors coming from that address will not be able to see the content covered by this directive. If, instead, you have a machine name, rather than an IP address, you can use that.

```
Deny from host.example.com
```

And, if you'd like to block access from an entire domain, you can specify just part of an address or domain name:

```
Deny from 192.168.205  
Deny from phishers.example.com moreidiots.example  
Deny from ke
```

Using [Order](#) will let you be sure that you are actually restricting things to the group that you want to let in, by combining a [Deny](#) and an [Allow](#) directive:

```
Order deny,allow
Deny from all
Allow from dev.example.com
```

Listing just the [Allow](#) directive would not do what you want, because it will let folks from that host in, in addition to letting everyone in. What you want is to let *only* those folks in.



Access Control by Environment Variable

`mod_authz_host`, in conjunction with `mod_setenvif`, can be used to restrict access to your website based on the value of arbitrary environment variables. This is done with the `Allow from env=` and `Deny from env=` syntax.

```
SetEnvIf User-Agent BadBot GoAway=1
Order allow,deny
Allow from all
Deny from env=GoAway
```

Warning:

Access control by User - Agent is an unreliable technique, since the User - Agent header can be set to anything at all, at the whim of the end user.

In the above example, the environment variable `GoAway` is set to `1` if the `User - Agent` matches the string `BadBot`. Then we deny access for any request when this variable is set. This blocks that particular user agent from the site.

An environment variable test can be negated using the `=!` syntax:

```
Allow from env=!GoAway
```



The [F] `RewriteRule` flag causes a 403 Forbidden response to be sent. Using this, you can deny access to a resource based on arbitrary criteria.

For example, if you wish to block access to a resource between 8pm and 6am, you can do this using `mod_rewrite`.

```
RewriteEngine On
RewriteCond %{TIME_HOUR} >20 [OR]
RewriteCond %{TIME_HOUR} <07
RewriteRule ^/fridge - [F]
```

This will return a 403 Forbidden response for any request after 8pm or before 7am. This technique can be used for any criteria that you wish to check. You can also redirect, or otherwise rewrite these requests, if that approach is preferred.



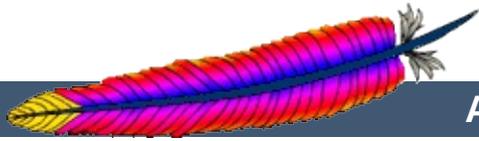
MORE INFORMATION

You should also read the documentation for [mod_auth_basic](#) and [mod_authz_host](#) which contain some more information about how this all works. [mod_authn_alias](#) can also help in simplifying certain authentication configurations.

See the [Authentication and Authorization](#) howto.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [How-To / Tutorials](#)

Apache Tutorial: Dynamic Content with CGI



Related Modules	Related Directives
-----------------	--------------------

mod_alias	AddHandler
mod_cgi	Options
mod_cgid	ScriptAlias

The CGI (Common Gateway Interface) defines a way for a web server to interact with external content-generating programs, which are often referred to as CGI programs or CGI scripts. It is the simplest, and most common, way to put dynamic content on your web site. This document will be an introduction to setting up CGI on your Apache web server, and getting started writing CGI programs.



Configuring Apache to permit CGI

In order to get your CGI programs to work properly, you'll need to have Apache configured to permit CGI execution. There are several ways to do this.

Note: If Apache has been built with shared module support you need to ensure that the module is loaded; in your `httpd.conf` you need to make sure the `LoadModule` directive has not been commented out. A correctly configured directive may look like this:

```
LoadModule cgid_module modules/mod_cgid.so
```

On Windows, or using a non-threaded MPM like `prefork`, A correctly configured directive may look like this:

```
LoadModule cgi_module modules/mod_cgi.so
```

ScriptAlias

The `ScriptAlias` directive tells Apache that a particular directory is set aside for CGI programs. Apache will assume that every file in this directory is a CGI program, and will attempt to execute it, when that particular resource is requested by a client.

The `ScriptAlias` directive looks like:

```
ScriptAlias /cgi-bin/ /usr/local/apache2/cgi-bin/
```

The example shown is from your default `httpd.conf` configuration file, if you installed Apache in the default location. The `ScriptAlias` directive is much like the `Alias` directive, which defines a URL prefix that is to mapped to a particular directory. `Alias` and `ScriptAlias` are usually used for directories that are outside of the `DocumentRoot` directory. The difference between `Alias` and `ScriptAlias` is that

ScriptAlias has the added meaning that everything under that URL prefix will be considered a CGI program. So, the example above tells Apache that any request for a resource beginning with `/cgi-bin/` should be served from the directory `/usr/local/apache2/cgi-bin/`, and should be treated as a CGI program.

For example, if the URL `http://www.example.com/cgi-bin/test.pl` is requested, Apache will attempt to execute the file `/usr/local/apache2/cgi-bin/test.pl` and return the output. Of course, the file will have to exist, and be executable, and return output in a particular way, or Apache will return an error message.

CGI outside of ScriptAlias directories

CGI programs are often restricted to **ScriptAlias**'ed directories for security reasons. In this way, administrators can tightly control who is allowed to use CGI programs. However, if the proper security precautions are taken, there is no reason why CGI programs cannot be run from arbitrary directories. For example, you may wish to let users have web content in their home directories with the **UserDir** directive. If they want to have their own CGI programs, but don't have access to the main `cgi-bin` directory, they will need to be able to run CGI programs elsewhere.

There are two steps to allowing CGI execution in an arbitrary directory. First, the `cgi-script` handler must be activated using the **AddHandler** or **SetHandler** directive. Second, `ExecCGI` must be specified in the **Options** directive.

Explicitly using Options to permit CGI execution

You could explicitly use the **Options** directive, inside your main server configuration file, to specify that CGI execution was permitted in a particular directory:

```
<Directory /usr/local/apache2/htdocs/somedir>
```

```
Options +ExecCGI
</Directory>
```

The above directive tells Apache to permit the execution of CGI files. You will also need to tell the server what files are CGI files. The following [AddHandler](#) directive tells the server to treat all files with the `cgi` or `pl` extension as CGI programs:

```
AddHandler cgi-script .cgi .pl
```

.htaccess files

The [.htaccess tutorial](#) shows how to activate CGI programs if you do not have access to `httpd.conf`.

User Directories

To allow CGI program execution for any file ending in `.cgi` in users' directories, you can use the following configuration.

```
<Directory /home/*/public_html>
  Options +ExecCGI
  AddHandler cgi-script .cgi
</Directory>
```

If you wish designate a `cgi-bin` subdirectory of a user's directory where everything will be treated as a CGI program, you can use the following.

```
<Directory /home/*/public_html/cgi-bin>
  Options ExecCGI
  SetHandler cgi-script
</Directory>
```



Writing a CGI program

There are two main differences between "regular" programming, and CGI programming.

First, all output from your CGI program must be preceded by a [MIME-type](#) header. This is HTTP header that tells the client what sort of content it is receiving. Most of the time, this will look like:

```
Content-type: text/html
```

Secondly, your output needs to be in HTML, or some other format that a browser will be able to display. Most of the time, this will be HTML, but occasionally you might write a CGI program that outputs a gif image, or other non-HTML content.

Apart from those two things, writing a CGI program will look a lot like any other program that you might write.

Your first CGI program

The following is an example CGI program that prints one line to your browser. Type in the following, save it to a file called `first.pl`, and put it in your `cgi-bin` directory.

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "Hello, World.";
```

Even if you are not familiar with Perl, you should be able to see what is happening here. The first line tells Apache (or whatever shell you happen to be running under) that this program can be executed by feeding the file to the interpreter found at the location `/usr/bin/perl`. The second line prints the content-type declaration we talked about, followed by two carriage-return newline pairs. This puts a blank line after the header, to indicate the end of the HTTP headers, and the beginning of the body. The third line prints the string "Hello, World.". And that's the end of it.

If you open your favorite browser and tell it to get the address

```
http://www.example.com/cgi-bin/first.pl
```

or wherever you put your file, you will see the one line `Hello, World.` appear in your browser window. It's not very exciting, but once you get that working, you'll have a good chance of getting just about anything working.



There are four basic things that you may see in your browser when you try to access your CGI program from the web:

The output of your CGI program

Great! That means everything worked fine. If the output is correct, but the browser is not processing it correctly, make sure you have the correct Content - Type set in your CGI program.

The source code of your CGI program or a "POST Method Not Allowed" message

That means that you have not properly configured Apache to process your CGI program. Reread the section on [configuring Apache](#) and try to find what you missed.

A message starting with "Forbidden"

That means that there is a permissions problem. Check the [Apache error log](#) and the section below on [file permissions](#).

A message saying "Internal Server Error"

If you check the [Apache error log](#), you will probably find that it says "Premature end of script headers", possibly along with an error message generated by your CGI program. In this case, you will want to check each of the below sections to see what might be preventing your CGI program from emitting the proper HTTP headers.

File permissions

Remember that the server does not run as you. That is, when the server starts up, it is running with the permissions of an unprivileged user - usually nobody, or www - and so it will need extra permissions to execute files that are owned by you. Usually, the way to give a file sufficient permissions to be executed by nobody is to give everyone execute permission on the file:

```
chmod a+x first.pl
```

Also, if your program reads from, or writes to, any other files, those files will need to have the correct permissions to permit this.

Path information and environment

When you run a program from your command line, you have certain information that is passed to the shell without you thinking about it. For example, you have a PATH, which tells the shell where it can look for files that you reference.

When a program runs through the web server as a CGI program, it may not have the same PATH. Any programs that you invoke in your CGI program (like `sendmail`, for example) will need to be specified by a full path, so that the shell can find them when it attempts to execute your CGI program.

A common manifestation of this is the path to the script interpreter (often `perl`) indicated in the first line of your CGI program, which will look something like:

```
#!/usr/bin/perl
```

Make sure that this is in fact the path to the interpreter.

When editing CGI scripts on Windows, end-of-line characters may be appended to the interpreter path. Ensure that files are then transferred to the server in ASCII mode. Failure to do so may result in "Command not found" warnings from the OS, due to the unrecognized end-of-line character being interpreted as a part of the interpreter filename.

Missing environment variables

If your CGI program depends on non-standard [environment variables](#), you will need to assure that those variables are passed by Apache.

When you miss HTTP headers from the environment, make sure they are formatted according to [RFC 2616](#), section 4.2: Header names must start with a letter, followed only by letters, numbers or hyphen. Any header violating this rule will be dropped silently.

Program errors

Most of the time when a CGI program fails, it's because of a problem with the program itself. This is particularly true once you get the hang of this CGI stuff, and no longer make the above two mistakes. The first thing to do is to make sure that your program runs from the command line before testing it via the web server. For example, try:

```
cd /usr/local/apache2/cgi-bin
./first.pl
```

(Do not call the perl interpreter. The shell and Apache should find the interpreter using the [path information](#) on the first line of the script.)

The first thing you see written by your program should be a set of HTTP headers, including the Content - Type, followed by a blank line. If you see anything else, Apache will return the Premature end of script headers error if you try to run it through the server. See [Writing a CGI program](#) above for more details.

Error logs

The error logs are your friend. Anything that goes wrong generates message in the error log. You should always look there first. If the place where you are hosting your web site does not permit you access to the error log, you should probably host your site somewhere else. Learn to read the error logs, and you'll find that almost all of your problems are quickly identified, and quickly solved.

Suexec

The [suexec](#) support program allows CGI programs to be run under different user permissions, depending on which virtual host or user home directory they are located in. Suexec has very strict permission checking, and any failure in that checking will result in your CGI programs failing with Premature end of script headers.

To check if you are using suexec, run `apachectl -V` and check for the location of SUEXEC_BIN. If Apache finds an [suexec](#) binary there on startup, suexec will be activated.

Unless you fully understand suexec, you should not be using it. To disable suexec, simply remove (or rename) the [suexec](#) binary pointed to by SUEXEC_BIN and then restart the server. If, after reading about [suexec](#), you still wish to use it, then run `suexec -V` to find the location of the suexec log file, and use that log file to find what policy you are violating.



As you become more advanced in CGI programming, it will become useful to understand more about what's happening behind the scenes. Specifically, how the browser and server communicate with one another. Because although it's all very well to write a program that prints "Hello, World.", it's not particularly useful.

Environment variables

Environment variables are values that float around you as you use your computer. They are useful things like your path (where the computer searches for the actual file implementing a command when you type it), your username, your terminal type, and so on. For a full list of your normal, every day environment variables, type `env` at a command prompt.

During the CGI transaction, the server and the browser also set environment variables, so that they can communicate with one another. These are things like the browser type (Netscape, IE, Lynx), the server type (Apache, IIS, WebSite), the name of the CGI program that is being run, and so on.

These variables are available to the CGI programmer, and are half of the story of the client-server communication. The complete list of required variables is at [Common Gateway Interface RFC](#).

This simple Perl CGI program will display all of the environment variables that are being passed around. Two similar programs are included in the `cgi-bin` directory of the Apache distribution. Note that some variables are required, while others are optional, so you may see some variables listed that were not in the official list. In addition, Apache provides many different ways for you to [add your own environment variables](#) to the basic ones provided by default.

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

```
foreach $key (keys %ENV) {  
    print "$key --> $ENV{$key}<br>";  
}
```

STDIN and STDOUT

Other communication between the server and the client happens over standard input (STDIN) and standard output (STDOUT). In normal everyday context, STDIN means the keyboard, or a file that a program is given to act on, and STDOUT usually means the console or screen.

When you POST a web form to a CGI program, the data in that form is bundled up into a special format and gets delivered to your CGI program over STDIN. The program then can process that data as though it was coming in from the keyboard, or from a file

The "special format" is very simple. A field name and its value are joined together with an equals (=) sign, and pairs of values are joined together with an ampersand (&). Inconvenient characters like spaces, ampersands, and equals signs, are converted into their hex equivalent so that they don't gum up the works. The whole data string might look something like:

```
name=Rich%20Bowen&city=Lexington&state=KY&sidekick=Squirrel%20Monkey
```

You'll sometimes also see this type of string appended to a URL. When that is done, the server puts that string into the environment variable called QUERY_STRING. That's called a GET request. Your HTML form specifies whether a GET or a POST is used to deliver the data, by setting the METHOD attribute in the FORM tag.

Your program is then responsible for splitting that string up into useful information. Fortunately, there are libraries and modules available to help you process this data, as well as handle other of the aspects of your CGI program.



When you write CGI programs, you should consider using a code library, or module, to do most of the grunt work for you. This leads to fewer errors, and faster development.

If you're writing CGI programs in Perl, modules are available on [CPAN](#). The most popular module for this purpose is `CGI.pm`. You might also consider `CGI::Lite`, which implements a minimal set of functionality, which is all you need in most programs.

If you're writing CGI programs in C, there are a variety of options. One of these is the CGIC library, from <http://www.boutell.com/cgic/>.



There are a large number of CGI resources on the web. The list from the HTML Writers Guild is a great source of answers to your questions. You can find out more at <http://www.hwg.org/lists/hwg-servers/>.

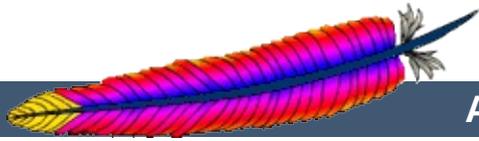
The current CGI specification is available in the [Common Gateway Interface RFC](#).

When you post a question about a CGI problem that you're having, whether to a mailing list, or to a newsgroup, make sure you provide enough information about what happened, what you expected to happen, and how what actually happened was different, what server you're running, what language your CGI program was in, and, if possible, the offending code. This will make finding your problem much simpler.

Note that questions about CGI problems should **never** be posted to the Apache bug database unless you are sure you have found a problem in the Apache source code.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [How-To / Tutorials](#)

Apache Tutorial: Introduction to Server Side Includes

Server-side includes provide a means to add dynamic content to existing HTML documents.



Related Modules	Related Directives
mod_include	Options
mod_cgi	XBitHack
mod_expires	AddType
	SetOutputFilter
	BrowserMatchNoCase

This article deals with Server Side Includes, usually called simply SSI. In this article, I'll talk about configuring your server to permit SSI, and introduce some basic SSI techniques for adding dynamic content to your existing HTML pages.

In the latter part of the article, we'll talk about some of the somewhat more advanced things that can be done with SSI, such as conditional statements in your SSI directives.



SSI (Server Side Includes) are directives that are placed in HTML pages, and evaluated on the server while the pages are being served. They let you add dynamically generated content to an existing HTML page, without having to serve the entire page via a CGI program, or other dynamic technology.

The decision of when to use SSI, and when to have your page entirely generated by some program, is usually a matter of how much of the page is static, and how much needs to be recalculated every time the page is served. SSI is a great way to add small pieces of information, such as the current time. But if a majority of your page is being generated at the time that it is served, you need to look for some other solution.



To permit SSI on your server, you must have the following directive either in your `httpd.conf` file, or in a `.htaccess` file:

```
Options +Includes
```

This tells Apache that you want to permit files to be parsed for SSI directives. Note that most configurations contain multiple `Options` directives that can override each other. You will probably need to apply the `Options` to the specific directory where you want SSI enabled in order to assure that it gets evaluated last.

Not just any file is parsed for SSI directives. You have to tell Apache which files should be parsed. There are two ways to do this. You can tell Apache to parse any file with a particular file extension, such as `.shtml`, with the following directives:

```
AddType text/html .shtml  
AddOutputFilter INCLUDES .shtml
```

One disadvantage to this approach is that if you wanted to add SSI directives to an existing page, you would have to change the name of that page, and all links to that page, in order to give it a `.shtml` extension, so that those directives would be executed.

The other method is to use the `XBitHack` directive:

```
XBitHack on
```

`XBitHack` tells Apache to parse files for SSI directives if they have the execute bit set. So, to add SSI directives to an existing page, rather than having to change the file name, you would just need to make the file executable using `chmod`.

```
chmod +x pagename.html
```

A brief comment about what not to do. You'll occasionally see people recommending that you just tell Apache to parse all `.html` files for SSI, so that you don't have to mess with `.shtml` file names. These folks have perhaps not heard about [XBitHack](#). The thing to keep in mind is that, by doing this, you're requiring that Apache read through every single file that it sends out to clients, even if they don't contain any SSI directives. This can slow things down quite a bit, and is not a good idea.

Of course, on Windows, there is no such thing as an execute bit to set, so that limits your options a little.

In its default configuration, Apache does not send the last modified date or content length HTTP headers on SSI pages, because these values are difficult to calculate for dynamic content. This can prevent your document from being cached, and result in slower perceived client performance. There are two ways to solve this:

1. Use the `XBitHack Full` configuration. This tells Apache to determine the last modified date by looking only at the date of the originally requested file, ignoring the modification date of any included files.
2. Use the directives provided by [mod_expires](#) to set an explicit expiration time on your files, thereby letting browsers and proxies know that it is acceptable to cache them.



SSI directives have the following syntax:

```
<!--#element attribute=value attribute=value ... -->
```

It is formatted like an HTML comment, so if you don't have SSI correctly enabled, the browser will ignore it, but it will still be visible in the HTML source. If you have SSI correctly configured, the directive will be replaced with its results.

The element can be one of a number of things, and we'll talk some more about most of these in the next installment of this series. For now, here are some examples of what you can do with SSI

Today's date

```
<!--#echo var="DATE_LOCAL" -->
```

The echo element just spits out the value of a variable. There are a number of standard variables, which include the whole set of environment variables that are available to CGI programs. Also, you can define your own variables with the set element.

If you don't like the format in which the date gets printed, you can use the config element, with a timefmt attribute, to modify that formatting.

```
<!--#config timefmt="%A %B %d, %Y" -->  
Today is <!--#echo var="DATE_LOCAL" -->
```

Modification date of the file

```
This document last modified <!--#flastmod file="index.html" -->
```

This element is also subject to timefmt format configurations.

Including the results of a CGI program

This is one of the more common uses of SSI - to output the results of a CGI program, such as everybody's favorite, a ``hit counter."

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```



Additional Examples

Following are some specific examples of things you can do in your HTML documents with SSI.

When was this document modified?

Earlier, we mentioned that you could use SSI to inform the user when the document was most recently modified. However, the actual method for doing that was left somewhat in question. The following code, placed in your HTML document, will put such a time stamp on your page. Of course, you will have to have SSI correctly enabled, as discussed above.

```
<!--#config timefmt="%A %B %d, %Y" -->
This file last modified <!--#flastmod file="ssi.shtml" -->
```

Of course, you will need to replace the `ssi.shtml` with the actual name of the file that you're referring to. This can be inconvenient if you're just looking for a generic piece of code that you can paste into any file, so you probably want to use the `LAST_MODIFIED` variable instead:

```
<!--#config timefmt="%D" -->
This file last modified <!--#echo var="LAST_MODIFIED" -->
```

For more details on the `timefmt` format, go to your favorite search site and look for `strftime`. The syntax is the same.

Including a standard footer

If you are managing any site that is more than a few pages, you may find that making changes to all those pages can be a real pain, particularly if you are trying to maintain some kind of standard look across all those pages.

Using an include file for a header and/or a footer can reduce the burden of these updates. You just have to make one footer file, and then include it into each page with the `include` SSI command. The `include`

element can determine what file to include with either the `file` attribute, or the `virtual` attribute. The `file` attribute is a file path, *relative to the current directory*. That means that it cannot be an absolute file path (starting with `/`), nor can it contain `../` as part of that path. The `virtual` attribute is probably more useful, and should specify a URL relative to the document being served. It can start with a `/`, but must be on the same server as the file being served.

```
<!--#include virtual="/footer.html" -->
```

I'll frequently combine the last two things, putting a `LAST_MODIFIED` directive inside a footer file to be included. SSI directives can be contained in the included file, and includes can be nested - that is, the included file can include another file, and so on.



What else can I config?

In addition to being able to config the time format, you can also config two other things.

Usually, when something goes wrong with your SSI directive, you get the message

```
[an error occurred while processing this directive]
```

If you want to change that message to something else, you can do so with the `errmsg` attribute to the `config` element:

```
<!--#config errmsg="[It appears that you don't know how to use SSI]" ->
```

Hopefully, end users will never see this message, because you will have resolved all the problems with your SSI directives before your site goes live. (Right?)

And you can config the format in which file sizes are returned with the `sizefmt` attribute. You can specify `bytes` for a full count in bytes, or `abbrev` for an abbreviated number in Kb or Mb, as appropriate.



Executing Commands

I expect that I'll have an article some time in the coming months about using SSI with small CGI programs. For now, here's something else that you can do with the exec element. You can actually have SSI execute a command using the shell (/bin/sh, to be precise - or the DOS shell, if you're on Win32). The following, for example, will give you a directory listing.

```
<pre>  
<!--#exec cmd="ls" -->  
</pre>
```

or, on Windows

```
<pre>  
<!--#exec cmd="dir" -->  
</pre>
```

You might notice some strange formatting with this directive on Windows, because the output from dir contains the string ``<dir>" in it, which confuses browsers.

Note that this feature is exceedingly dangerous, as it will execute whatever code happens to be embedded in the exec tag. If you have any situation where users can edit content on your web pages, such as with a ``guestbook", for example, make sure that you have this feature disabled. You can allow SSI, but not the exec feature, with the IncludesNOEXEC argument to the Options directive.



In addition to spitting out content, Apache SSI gives you the option of setting variables, and using those variables in comparisons and conditionals.

Setting variables

Using the `set` directive, you can set variables for later use. We'll need this later in the discussion, so we'll talk about it here. The syntax of this is as follows:

```
<!--#set var="name" value="Rich" -->
```

In addition to merely setting values literally like that, you can use any other variable, including [environment variables](#) or the variables discussed above (like `LAST_MODIFIED`, for example) to give values to your variables. You will specify that something is a variable, rather than a literal string, by using the dollar sign (\$) before the name of the variable.

```
<!--#set var="modified" value="$LAST_MODIFIED" -->
```

To put a literal dollar sign into the value of your variable, you need to escape the dollar sign with a backslash.

```
<!--#set var="cost" value="\$100" -->
```

Finally, if you want to put a variable in the midst of a longer string, and there's a chance that the name of the variable will run up against some other characters, and thus be confused with those characters, you can place the name of the variable in braces, to remove this confusion. (It's hard to come up with a really good example of this, but hopefully you'll get the point.)

```
<!--#set var="date" value="{DATE_LOCAL}_{DATE_GMT}" -->
```

Conditional expressions

Now that we have variables, and are able to set and compare their values, we can use them to express conditionals. This lets SSI be a tiny programming language of sorts. [mod_include](#) provides an `if`, `elif`, `else`, `endif` structure for building conditional statements. This allows you to effectively generate multiple logical pages out of one actual page.

The structure of this conditional construct is:

```
<!--#if expr="test_condition" -->
<!--#elif expr="test_condition" -->
<!--#else -->
<!--#endif -->
```

A *test_condition* can be any sort of logical comparison - either comparing values to one another, or testing the "truth" of a particular value. (A given string is true if it is nonempty.) For a full list of the comparison operators available to you, see the [mod_include](#) documentation. Here are some examples of how one might use this construct.

In your configuration file, you could put the following line:

```
BrowserMatchNoCase macintosh Mac
BrowserMatchNoCase MSIE InternetExplorer
```

This will set environment variables "Mac" and "InternetExplorer" to true, if the client is running Internet Explorer on a Macintosh.

Then, in your SSI-enabled document, you might do the following:

```
<!--#if expr="${Mac} && ${InternetExplorer}" -->
Apologetic text goes here
<!--#else -->
Cool JavaScript code goes here
<!--#endif -->
```

Not that I have anything against IE on Macs - I just struggled for a few

hours last week trying to get some JavaScript working on IE on a Mac, when it was working everywhere else. The above was the interim workaround.

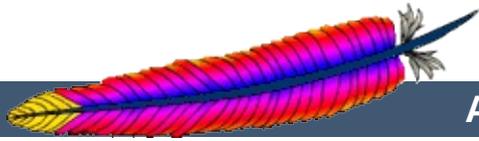
Any other variable (either ones that you define, or normal environment variables) can be used in conditional statements. With Apache's ability to set environment variables with the `SetEnvIf` directives, and other related directives, this functionality can let you do some pretty involved dynamic stuff without ever resorting to CGI.



SSI is certainly not a replacement for CGI, or other technologies used for generating dynamic web pages. But it is a great way to add small amounts of dynamic content to pages, without doing a lot of extra work.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [How-To / Tutorials](#)

Apache HTTP Server Tutorial: .htaccess files

.htaccess files provide a way to make configuration changes on a per-directory basis.



Related Modules	Related Directives
core	AccessFileName
mod_authn_file	AllowOverride
mod_authz_groupfile	Options
mod_cgi	AddHandler
mod_include	SetHandler
mod_mime	AuthType
	AuthName
	AuthUserFile
	AuthGroupFile
	Require

You should avoid using .htaccess files completely if you have access to httpd main server config file. Using .htaccess files slows down your Apache http server. Any directive that you can include in a .htaccess file is better set in a [Directory](#) block, as it will have the same effect with better performance.



.htaccess files (or "distributed configuration files") provide a way to make configuration changes on a per-directory basis. A file, containing one or more configuration directives, is placed in a particular document directory, and the directives apply to that directory, and all subdirectories thereof.

Note:

If you want to call your .htaccess file something else, you can change the name of the file using the [AccessFileName](#) directive. For example, if you would rather call the file .config then you can put the following in your server configuration file:

```
AccessFileName .config
```

In general, .htaccess files use the same syntax as the [main configuration files](#). What you can put in these files is determined by the [AllowOverride](#) directive. This directive specifies, in categories, what directives will be honored if they are found in a .htaccess file. If a directive is permitted in a .htaccess file, the documentation for that directive will contain an [Override](#) section, specifying what value must be in [AllowOverride](#) in order for that directive to be permitted.

For example, if you look at the documentation for the [AddDefaultCharset](#) directive, you will find that it is permitted in .htaccess files. (See the Context line in the directive summary.) The [Override](#) line reads FileInfo. Thus, you must have at least `AllowOverride FileInfo` in order for this directive to be honored in .htaccess files.

Example:

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

If you are unsure whether a particular directive is permitted in a .htaccess file, look at the documentation for that directive, and check the Context line for ".htaccess".



In general, you should only use `.htaccess` files when you don't have access to the main server configuration file. There is, for example, a common misconception that user authentication should always be done in `.htaccess` files, and, in more recent years, another misconception that `mod_rewrite` directives must go in `.htaccess` files. This is simply not the case. You can put user authentication configurations in the main server configuration, and this is, in fact, the preferred way to do things. Likewise, `mod_rewrite` directives work better, in many respects, in the main server configuration.

`.htaccess` files should be used in a case where the content providers need to make configuration changes to the server on a per-directory basis, but do not have root access on the server system. In the event that the server administrator is not willing to make frequent configuration changes, it might be desirable to permit individual users to make these changes in `.htaccess` files for themselves. This is particularly true, for example, in cases where ISPs are hosting multiple user sites on a single machine, and want their users to be able to alter their configuration.

However, in general, use of `.htaccess` files should be avoided when possible. Any configuration that you would consider putting in a `.htaccess` file, can just as effectively be made in a `<Directory>` section in your main server configuration file.

There are two main reasons to avoid the use of `.htaccess` files.

The first of these is performance. When `AllowOverride` is set to allow the use of `.htaccess` files, `httpd` will look in every directory for `.htaccess` files. Thus, permitting `.htaccess` files causes a performance hit, whether or not you actually even use them! Also, the `.htaccess` file is loaded every time a document is requested.

Further note that `httpd` must look for `.htaccess` files in all higher-level

directories, in order to have a full complement of directives that it must apply. (See section on [how directives are applied](#).) Thus, if a file is requested out of a directory `/www/htdocs/example`, `httpd` must look for the following files:

```
/.htaccess
/www/.htaccess
/www/htdocs/.htaccess
/www/htdocs/example/.htaccess
```

And so, for each file access out of that directory, there are 4 additional file-system accesses, even if none of those files are present. (Note that this would only be the case if `.htaccess` files were enabled for `/`, which is not usually the case.)

In the case of [RewriteRule](#) directives, in `.htaccess` context these regular expressions must be re-compiled with every request to the directory, whereas in main server configuration context they are compiled once and cached. Additionally, the rules themselves are more complicated, as one must work around the restrictions that come with per-directory context and `mod_rewrite`. Consult the [Rewrite Guide](#) for more detail on this subject.

The second consideration is one of security. You are permitting users to modify server configuration, which may result in changes over which you have no control. Carefully consider whether you want to give your users this privilege. Note also that giving users less privileges than they need will lead to additional technical support requests. Make sure you clearly tell your users what level of privileges you have given them. Specifying exactly what you have set [AllowOverride](#) to, and pointing them to the relevant documentation, will save yourself a lot of confusion later.

Note that it is completely equivalent to put a `.htaccess` file in a directory `/www/htdocs/example` containing a directive, and to put that same directive in a `Directory` section `<Directory`

/www/htdocs/example> in your main server configuration:

.htaccess file in /www/htdocs/example:

Contents of .htaccess file in /www/htdocs/example

```
AddType text/example .exm
```

Section from your httpd.conf file

```
<Directory /www/htdocs/example>  
    AddType text/example .exm  
</Directory>
```

However, putting this configuration in your server configuration file will result in less of a performance hit, as the configuration is loaded once when httpd starts, rather than every time a file is requested.

The use of .htaccess files can be disabled completely by setting the [AllowOverride](#) directive to none:

```
AllowOverride None
```



How directives are applied

The configuration directives found in a `.htaccess` file are applied to the directory in which the `.htaccess` file is found, and to all subdirectories thereof. However, it is important to also remember that there may have been `.htaccess` files in directories higher up. Directives are applied in the order that they are found. Therefore, a `.htaccess` file in a particular directory may override directives found in `.htaccess` files found higher up in the directory tree. And those, in turn, may have overridden directives found yet higher up, or in the main server configuration file itself.

Example:

In the directory `/www/htdocs/example1` we have a `.htaccess` file containing the following:

```
Options +ExecCGI
```

(Note: you must have "AllowOverride Options" in effect to permit the use of the "[Options](#)" directive in `.htaccess` files.)

In the directory `/www/htdocs/example1/example2` we have a `.htaccess` file containing:

```
Options Includes
```

Because of this second `.htaccess` file, in the directory `/www/htdocs/example1/example2`, CGI execution is not permitted, as only `Options Includes` is in effect, which completely overrides any earlier setting that may have been in place.

Merging of `.htaccess` with the main configuration files

As discussed in the documentation on [Configuration Sections](#),

.htaccess files can override the [<Directory>](#) sections for the corresponding directory, but will be overridden by other types of configuration sections from the main configuration files. This fact can be used to enforce certain configurations, even in the presence of a liberal [AllowOverride](#) setting. For example, to prevent script execution while allowing anything else to be set in .htaccess you can use:

```
<Directory />
  Allowoverride All
</Directory>

<Location />
  Options +IncludesNoExec -ExecCGI
</Location>
```



If you jumped directly to this part of the document to find out how to do authentication, it is important to note one thing. There is a common misconception that you are required to use `.htaccess` files in order to implement password authentication. This is not the case. Putting authentication directives in a `<Directory>` section, in your main server configuration file, is the preferred way to implement this, and `.htaccess` files should be used only if you don't have access to the main server configuration file. See [above](#) for a discussion of when you should and should not use `.htaccess` files.

Having said that, if you still think you need to use a `.htaccess` file, you may find that a configuration such as what follows may work for you.

`.htaccess` file contents:

```
AuthType Basic
AuthName "Password Required"
AuthUserFile /www/passwords/password.file
AuthGroupFile /www/passwords/group.file
Require Group admins
```

Note that `AllowOverride AuthConfig` must be in effect for these directives to have any effect.

Please see the [authentication tutorial](#) for a more complete discussion of authentication and authorization.



Server Side Includes Example

Another common use of `.htaccess` files is to enable Server Side Includes for a particular directory. This may be done with the following configuration directives, placed in a `.htaccess` file in the desired directory:

```
Options +Includes
AddType text/html shtml
AddHandler server-parsed shtml
```

Note that `AllowOverride Options` and `AllowOverride FileInfo` must both be in effect for these directives to have any effect.

Please see the [SSI tutorial](#) for a more complete discussion of server-side includes.



Finally, you may wish to use a `.htaccess` file to permit the execution of CGI programs in a particular directory. This may be implemented with the following configuration:

```
Options +ExecCGI
AddHandler cgi-script cgi pl
```

Alternately, if you wish to have all files in the given directory be considered to be CGI programs, this may be done with the following configuration:

```
Options +ExecCGI
SetHandler cgi-script
```

Note that `AllowOverride Options` and `AllowOverride FileInfo` must both be in effect for these directives to have any effect.

Please see the [CGI tutorial](#) for a more complete discussion of CGI programming and configuration.



Troubleshooting

When you put configuration directives in a `.htaccess` file, and you don't get the desired effect, there are a number of things that may be going wrong.

Most commonly, the problem is that `AllowOverride` is not set such that your configuration directives are being honored. Make sure that you don't have a `AllowOverride None` in effect for the file scope in question. A good test for this is to put garbage in your `.htaccess` file and reload the page. If a server error is not generated, then you almost certainly have `AllowOverride None` in effect.

If, on the other hand, you are getting server errors when trying to access documents, check your `httpd` error log. It will likely tell you that the directive used in your `.htaccess` file is not permitted.

```
[Fri Sep 17 18:43:16 2010] [alert] [client 192.168.200.51]
/var/www/html/.htaccess: DirectoryIndex not allowed here
```

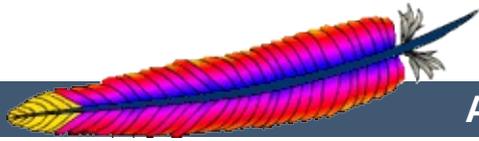
This will indicate either that you've used a directive that is never permitted in `.htaccess` files, or that you simply don't have `AllowOverride` set to a level sufficient for the directive you've used. Consult the documentation for that particular directive to determine which is the case.

Alternately, it may tell you that you had a syntax error in your usage of the directive itself.

```
[Sat Aug 09 16:22:34 2008] [alert] [client 192.168.200.51]
/var/www/html/.htaccess: RewriteCond: bad flag delimiters
```

In this case, the error message should be specific to the particular syntax error that you have committed.

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [How-To / Tutorials](#)

Per-user web directories

On systems with multiple users, each user can be permitted to have a web site in their home directory using the `UserDir` directive. Visitors to a URL `http://example.com/~username/` will get content out of the home directory of the user "username", out of the subdirectory specified by the `UserDir` directive.

Note that, by default, access to these directories is **not** enabled. You can enable access when using `UserDir` by uncommenting the line

```
#Include conf/extra/httpd-userdir.conf
```

in the default config file `conf/httpd.conf`, and adapting the `httpd-userdir.conf` file as necessary, or by including the appropriate directives in a `Directory` block within the main config file.

See also

[Mapping URLs to the Filesystem](#)



Related Modules

[mod_userdir](#)

Related Directives

[UserDir](#)

[DirectoryMatch](#)

[AllowOverride](#)



The `UserDir` directive specifies a directory out of which per-user content is loaded. This directive may take several different forms.

If a path is given which does not start with a leading slash, it is assumed to be a directory path relative to the home directory of the specified user. Given this configuration:

```
UserDir public_html
```

the URL `http://example.com/~rbowen/file.html` will be translated to the file path `/home/rbowen/public_html/file.html`

If a path is given starting with a slash, a directory path will be constructed using that path, plus the username specified. Given this configuration:

```
UserDir /var/html
```

the URL `http://example.com/~rbowen/file.html` will be translated to the file path `/var/html/rbowen/file.html`

If a path is provided which contains an asterisk (*), a path is used in which the asterisk is replaced with the username. Given this configuration:

```
UserDir /var/www/*/docs
```

the URL `http://example.com/~rbowen/file.html` will be translated to the file path `/var/www/rbowen/docs/file.html`

Multiple directories or directory paths can also be set.

```
UserDir public_html /var/html
```

For the URL `http://example.com/~rbowen/file.html`, Apache

will search for ~rbowen. If it isn't found, Apache will search for rbowen in /var/html. If found, the above URL will then be translated to the file path /var/html/rbowen/file.html



Redirecting to External URLs

The `UserDir` directive can be used to redirect user directory requests to external URLs.

```
UserDir http://example.org/users/**
```

The above example will redirect a request for `http://example.com/~bob/abc.html` to `http://example.org/users/bob/abc.html`.



Restricting what users are permitted to use the feature

Using the syntax shown in the UserDir documentation, you can restrict what users are permitted to use this functionality:

```
UserDir disabled root jro fish
```

The configuration above will enable the feature for all users except for those listed in the `disabled` statement. You can, likewise, disable the feature for all but a few users by using a configuration like the following:

```
UserDir disabled  
UserDir enabled rbowen krietz
```

See [UserDir](#) documentation for additional examples.



Enabling cgi directory for each user

In order to give each user their own cgi-bin directory, you can use a `<Directory>` directive to make a particular subdirectory of a user's home directory cgi-enabled.

```
<Directory /home/*/public_html/cgi-bin/>  
Options ExecCGI  
SetHandler cgi-script  
</Directory>
```

Then, presuming that `UserDir` is set to `public_html`, a cgi program `example.cgi` could be loaded from that directory as:

```
http://example.com/~rbowen/cgi-bin/example.cgi
```

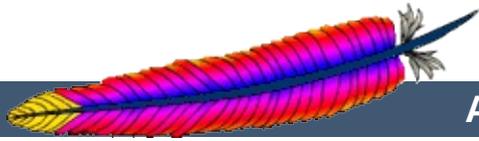


Allowing users to alter configuration

If you want to allow users to modify the server configuration in their web space, they will need to use `.htaccess` files to make these changes. Ensure that you have set `AllowOverride` to a value sufficient for the directives that you want to permit the users to modify. See the [.htaccess tutorial](#) for additional details on how this works.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Platform Specific Notes](#)

Using Apache HTTP Server on Microsoft Windows

This document explains how to install, configure and run Apache 2.2 under Microsoft Windows. If you have questions after reviewing the documentation (and any event and error logs), you should consult the peer-supported [users' mailing list](#).

This document assumes that you are installing a binary distribution of Apache. If you want to compile Apache yourself (possibly to help with development or tracking down bugs), see [Compiling Apache for Microsoft Windows](#).



Operating System Requirements

The primary Windows platform for running Apache 2.2 is Windows 2000 or later. The binary installer only works with the x86 family of processors, such as Intel and AMD processors. Always obtain and install the current service pack to avoid operating system bugs.

Running Apache on Windows 9x is ignored by the developers, and is strongly discouraged. On Windows NT 4.0, installing Service Pack 6 is required. Apache HTTP Server versions later than 2.2 will not run on any operating system earlier than Windows 2000.



Deploying Apache for Windows

The Apache HTTP Server Project itself does not provide binary releases of software, only source code. Individual committers *may* provide binary packages as a convenience, but it is not a release deliverable.

If you cannot compile the Apache HTTP Server yourself, you can obtain a binary package from numerous binary distributions available on the Internet.

Popular options for deploying Apache httpd, and, optionally, PHP and MySQL, on Microsoft Windows, include:

- [ApacheHaus](#)
- [Apache Lounge](#)
- [BitNami WAMP Stack](#)
- [WampServer](#)
- [XAMPP](#)



Installing Apache for Windows

You need Microsoft Installer 2.0 or above for the installation to work. For Windows NT 4.0 and 2000 refer to Microsoft's article [KB 292539](#). Windows XP and later do not require this update. The Windows 98/ME installer engine appears to no longer be available from Microsoft, and these instructions no longer detail such prerequisites.

Note that you cannot install two versions of Apache 2.2 on the same computer with the binary installer. You can, however, install a version of the 1.3 series **and** a version of the 2.2 series on the same computer without problems. If you need to have two different 2.2 versions on the same computer, you have to [compile and install Apache from the source](#).

Run the Apache `.msi` file you downloaded above. The installation will ask you for these things:

1. **Network Domain.** Enter the DNS domain in which your server is or will be registered in. For example, if your server's full DNS name is `server.mydomain.net`, you would type `mydomain.net` here.
2. **Server Name.** Your server's full DNS name. From the example above, you would type `server.mydomain.net` here.
3. **Administrator's Email Address.** Enter the server administrator's or webmaster's email address here. This address will be displayed along with error messages to the client by default.
4. **For whom to install Apache** Select for `All Users`, on Port `80`, as a `Service - Recommended` if you'd like your new Apache to listen at port 80 for incoming traffic. It will run as a service (that is, Apache will run even if no one is logged in on the server at the moment) Select `only for the Current User`, on Port `8080`, when started `Manually` if you'd like to install Apache for your personal experimenting or if you already have another WWW server running on port 80.

5. **The installation type.** Select `Typical` for everything except the source code and libraries for module development. With `Custom` you can specify what to install. A full install will require about 13 megabytes of free disk space. This does *not* include the size of your web site(s).
6. **Where to install.** The default path is `C:\Program Files\Apache Software Foundation` under which a directory called `Apache2.2` will be created by default.

During the installation, Apache will configure the files in the `conf` subdirectory to reflect the chosen installation directory. However, if any of the configuration files in this directory already exist, they will not be overwritten. Instead, the new copy of the corresponding file will be left with the extension `.default`. So, for example, if `conf\httpd.conf` already exists, it will be renamed as `conf\httpd.conf.default`. After the installation you should manually check to see what new settings are in the `.default` file, and if necessary, update your existing configuration file.

Also, if you already have a file called `htdocs\index.html`, it will not be overwritten (and no `index.html.default` will be installed either). This means it should be safe to install Apache over an existing installation, although you would have to stop the existing running server before doing the installation, and then start the new one after the installation is finished.

After installing Apache, you must edit the configuration files in the `conf` subdirectory as required. These files will be configured during the installation so that Apache is ready to be run from the directory it was installed into, with the documents server from the subdirectory `htdocs`. There are lots of other options which you should set before you really start using Apache. However, to get started quickly, the files should work as installed.



One feature of the installer, "Build Headers and Libraries", can be ignored by most users, but should be installed if compiling third party modules. The "APR Iconv Code Pages" can similarly be omitted by most users, unless using [mod_charset_lite](#) or a third party module which relies on APR internationalization features.

There are requests to ship a .zip file from time to time. There is no point in the ASF consuming additional storage, mirroring and bandwidth for this purpose, because the .msi installer allows the installation artifacts to all be unpacked using the `msiexec /a network` installation option. Using this command against any .msi will result in an exploded tree of all of the individual files and components.

The installation options above can be customized by users familiar with `msiexec.exe` options and silent installation. The actual installer sources are available in the `httpd/httpd/win32-msi/` tree of the `httpd` project subversion repository. For reference, some of the more common variables which may be modified are;

- `AgreeToLicense` (toggle to "Yes")
- `ALLUSERS` (choose between Null and "1")
- `ApplicationUsers` (toggle to "OnlyCurrentUser")
- `EXISTING_APACHE_SERVICE_PATH`
- `INSTALLDIR` (default "ProgramFilesFolder\Apache Software Foundation\Apache2.2\")
- `INSTALLLEVEL` (default "100", refer to list below)
- `RESOLVED_WINSOCK2` (toggle to "1")
- `SERVERADMIN`
- `SERVERDOMAIN`
- `SERVERNAME`
- `SERVERPORT` (default "80")
- `SERVERSSLPORT` (default "443")
- `SERVICEINTERNALNAME` (default "Apache2.2", no spaces!)

- SERVICENAME (default "Apache2.2", include spaces)
- SetupType (default "Typical")

The installation level of various features, which may be individually toggled, include;

- Apache (1, base Apache HTTP Server 2.2 feature)
- ApacheDocs (11, Apache Documentation)
- ApacheMonitoring (41, Apache Service Taskbar Icon)
- ApacheRuntime (1, Apache Runtime)
- BuildFiles (101, Build Headers and Libraries)
- Iconv (21, APR Iconv Code Pages)
- OpensslBin (31, OpenSSL Runtime)
- SslBin (41, Ssl Binaries)



Configuring Apache for Windows

Apache is configured by the files in the `conf` subdirectory. These are the same files used to configure the Unix version, but there are a few different directives for Apache on Windows. See the [directive index](#) for all the available directives.

The main differences in Apache for Windows are:

- Because Apache for Windows is multithreaded, it does not use a separate process for each request, as Apache can on Unix. Instead there are usually only two Apache processes running: a parent process, and a child which handles the requests. Within the child process each request is handled by a separate thread.

The process management directives are also different:

MaxRequestsPerChild: Like the Unix directive, this controls how many requests (actually, connections) which a single child process will serve before exiting. However, unlike on Unix, a replacement process is not instantly available. Use the default `MaxRequestsPerChild 0`, unless instructed to change the behavior to overcome a memory leak in third party modules or in-process applications.

Warning: The server configuration file is reread when a new child process is started. If you have modified `httpd.conf`, the new child may not start or you may receive unexpected results.

ThreadsPerChild: This directive is new. It tells the server how many threads it should use. This is the maximum number of connections the server can handle at once, so be sure to set this number high enough for your site if you get a lot of hits. The recommended default is `ThreadsPerChild 150`, but this must be

adjusted to reflect the greatest anticipated number of simultaneous connections to accept.

- The directives that accept filenames as arguments must use Windows filenames instead of Unix ones. However, because Apache may interpret backslashes as an "escape character" sequence, you should consistently use forward slashes in path names, not backslashes.
- While filenames are generally case-insensitive on Windows, URLs are still treated internally as case-sensitive before they are mapped to the filesystem. For example, the `<Location>`, `Alias`, and `ProxyPass` directives all use case-sensitive arguments. For this reason, it is particularly important to use the `<Directory>` directive when attempting to limit access to content in the filesystem, since this directive applies to any content in a directory, regardless of how it is accessed. If you wish to assure that only lowercase is used in URLs, you can use something like:

```
RewriteEngine On
RewriteMap lowercase int:tolower
RewriteCond %{REQUEST_URI} [A-Z]
RewriteRule (.*) ${lowercase:$1} [R,L]
```

- When running, Apache needs write access only to the logs directory and any configured cache directory tree. Due to the issue of case insensitive and short 8.3 format names, Apache must validate all path names given. This means that each directory which Apache evaluates, from the drive root up to the directory leaf, must have read, list and traverse directory permissions. If Apache2.2 is installed at C:\Program Files, then the root directory, Program Files and Apache2.2 must all be visible to Apache.
- Apache for Windows contains the ability to load modules at runtime, without recompiling the server. If Apache is compiled normally, it will

install a number of optional modules in the `\Apache2.2\modules` directory. To activate these or other modules, the new [LoadModule](#) directive must be used. For example, to activate the status module, use the following (in addition to the status-activating directives in `access.conf`):

```
LoadModule status_module modules/mod_status.so
```

Information on [creating loadable modules](#) is also available.

- Apache can also load ISAPI (Internet Server Application Programming Interface) extensions such as those used by Microsoft IIS and other Windows servers. [More information is available](#). Note that Apache **cannot** load ISAPI Filters, and ISAPI Handlers with some Microsoft feature extensions will not work.
- When running CGI scripts, the method Apache uses to find the interpreter for the script is configurable using the [ScriptInterpreterSource](#) directive.
- Since it is often difficult to manage files with names like `.htaccess` in Windows, you may find it useful to change the name of this per-directory configuration file using the [AccessFilename](#) directive.
- Any errors during Apache startup are logged into the Windows event log when running on Windows NT. This mechanism acts as a backup for those situations where Apache is not yet prepared to use the `error.log` file. You can review the Windows Application Event Log by using the Event Viewer, e.g. Start - Settings - Control Panel - Administrative Tools - Event Viewer.



Running Apache as a Service

You can install Apache as a service automatically during the installation. If you chose to install for all users, the installation will create an Apache service for you. If you specify to install for yourself only, you can manually register Apache as a service after the installation. You have to be a member of the Administrators group for the service installation to succeed.

Apache comes with a utility called the Apache Service Monitor. With it you can see and manage the state of all installed Apache services on any machine on your network. To be able to manage an Apache service with the monitor, you have to first install the service (either automatically via the installation or manually).

You can install Apache as a Windows NT service as follows from the command prompt at the Apache `bin` subdirectory:

```
httpd.exe -k install
```

If you need to specify the name of the service you want to install, use the following command. You have to do this if you have several different service installations of Apache on your computer. If you specify a name during the install, you have to also specify it during any other `-k` operation.

```
httpd.exe -k install -n "MyServiceName"
```

If you need to have specifically named configuration files for different services, you must use this:

```
httpd.exe -k install -n "MyServiceName" -f "c:\files\my.conf"
```

If you use the first command without any special parameters except `-k install`, the service will be called `Apache2.2` and the configuration will

be assumed to be `conf\httpd.conf`.

Removing an Apache service is easy. Just use:

```
httpd.exe -k uninstall
```

The specific Apache service to be uninstalled can be specified by using:

```
httpd.exe -k uninstall -n "MyServiceName"
```

Normal starting, restarting and shutting down of an Apache service is usually done via the Apache Service Monitor, by using commands like `NET START Apache2.2` and `NET STOP Apache2.2` or via normal Windows service management. Before starting Apache as a service by any means, you should test the service's configuration file by using:

```
httpd.exe -n "MyServiceName" -t
```

You can control an Apache service by its command line switches, too. To start an installed Apache service you'll use this:

```
httpd.exe -k start -n "MyServiceName"
```

To stop an Apache service via the command line switches, use this:

```
httpd.exe -k stop -n "MyServiceName"
```

or

```
httpd.exe -k shutdown -n "MyServiceName"
```

You can also restart a running service and force it to reread its configuration file by using:

```
httpd.exe -k restart -n "MyServiceName"
```

By default, all Apache services are registered to run as the system user (the LocalSystem account). The LocalSystem account has no privileges to your network via any Windows-secured mechanism, including the file system, named pipes, DCOM, or secure RPC. It has, however, wide privileges locally.

Never grant any network privileges to the LocalSystem account! If you need Apache to be able to access network resources, create a separate account for Apache as noted below.

It is recommended that users create a separate account for running Apache service(s). If you have to access network resources via Apache, this is required.

1. Create a normal domain user account, and be sure to memorize its password.
2. Grant the newly-created user a privilege of Log on as a service and Act as part of the operating system. On Windows NT 4.0 these privileges are granted via User Manager for Domains, but on Windows 2000 and XP you probably want to use Group Policy for propagating these settings. You can also manually set these via the Local Security Policy MMC snap-in.
3. Confirm that the created account is a member of the Users group.
4. Grant the account read and execute (RX) rights to all document and script folders (htdocs and cgi-bin for example).
5. Grant the account change (RWXD) rights to the Apache logs directory.
6. Grant the account read and execute (RX) rights to the httpd.exe binary executable.

It is usually a good practice to grant the user the Apache service runs as read and execute (RX) access to the whole Apache2.2 directory, except the logs subdirectory, where the user has to have at least change (RWXD) rights.

If you allow the account to log in as a user and as a service, then you can log on with that account and test that the account has the privileges to execute the scripts, read the web pages, and that you can start Apache in a console window. If this works, and you have followed the steps above, Apache should execute as a service with no problems.

Error code 2186 is a good indication that you need to review the "Log On As" configuration for the service, since Apache cannot access a required network resource. Also, pay close attention to the privileges of the user Apache is configured to run as.

When starting Apache as a service you may encounter an error message from the Windows Service Control Manager. For example, if you try to start Apache by using the Services applet in the Windows Control Panel, you may get the following message:

```
Could not start the Apache2.2 service on \\COMPUTER
Error 1067; The process terminated unexpectedly.
```

You will get this generic error if there is any problem with starting the Apache service. In order to see what is really causing the problem you should follow the instructions for Running Apache for Windows from the Command Prompt.

If you are having problems with the service, it is suggested you follow the instructions below to try starting httpd.exe from a console window, and work out the errors before struggling to start it as a service again.



Running Apache as a Console Application

Running Apache as a service is usually the recommended way to use it, but it is sometimes easier to work from the command line (on Windows 9x running Apache from the command line is the recommended way due to the lack of reliable service support.)

To run Apache from the command line as a console application, use the following command:

```
httpd.exe
```

Apache will execute, and will remain running until it is stopped by pressing Control-C.

You can also run Apache via the shortcut Start Apache in Console placed to Start Menu --> Programs --> Apache HTTP Server 2.2.xx --> Control Apache Server during the installation. This will open a console window and start Apache inside it. If you don't have Apache installed as a service, the window will remain visible until you stop Apache by pressing Control-C in the console window where Apache is running in. The server will exit in a few seconds. However, if you do have Apache installed as a service, the shortcut starts the service. If the Apache service is running already, the shortcut doesn't do anything.

If Apache is running as a service, you can tell it to stop by opening another console window and entering:

```
httpd.exe -k shutdown
```

Running as a service should be preferred over running in a console window because this lets Apache end any current operations and clean up gracefully.

But if the server is running in a console window, you can only stop it by pressing Control-C in the same window.

You can also tell Apache to restart. This forces it to reread the configuration file. Any operations in progress are allowed to complete without interruption. To restart Apache, either press Control-Break in the console window you used for starting Apache, or enter

```
httpd.exe -k restart
```

if the server is running as a service.

Note for people familiar with the Unix version of Apache: these commands provide a Windows equivalent to `kill -TERM pid` and `kill -USR1 pid`. The command line option used, `-k`, was chosen as a reminder of the `kill` command used on Unix.

If the Apache console window closes immediately or unexpectedly after startup, open the Command Prompt from the Start Menu --> Programs. Change to the folder to which you installed Apache, type the command `httpd.exe`, and read the error message. Then change to the logs folder, and review the `error.log` file for configuration mistakes. If you accepted the defaults when you installed Apache, the commands would be:

```
c:  
cd "\\Program Files\\Apache Software Foundation\\Apache2.2\\bin"  
httpd.exe
```

Then wait for Apache to stop, or press Control-C. Then enter the following:

```
cd ..\logs  
more < error.log
```

When working with Apache it is important to know how it will find the configuration file. You can specify a configuration file on the command line in two ways:

- -f specifies an absolute or relative path to a particular configuration file:

```
httpd.exe -f "c:\my server files\anotherconfig.conf"
```

or

```
httpd.exe -f files\anotherconfig.conf
```

- -n specifies the installed Apache service whose configuration file is to be used:

```
httpd.exe -n "MyServiceName"
```

In both of these cases, the proper [ServerRoot](#) should be set in the configuration file.

If you don't specify a configuration file with -f or -n, Apache will use the file name compiled into the server, such as conf\httpd.conf. This built-in path is relative to the installation directory. You can verify the compiled file name from a value labelled as SERVER_CONFIG_FILE when invoking Apache with the -V switch, like this:

```
httpd.exe -V
```

Apache will then try to determine its [ServerRoot](#) by trying the following, in this order:

1. A [ServerRoot](#) directive via the -C command line switch.
2. The -d switch on the command line.
3. Current working directory.
4. A registry entry which was created if you did a binary installation.

5. The server root compiled into the server. This is /apache by default, you can verify it by using `httpd.exe -V` and looking for a value labelled as `HTTPD_ROOT`.

During the installation, a version-specific registry key is created in the Windows registry. The location of this key depends on the type of the installation. If you chose to install Apache for all users, the key is located under the `HKEY_LOCAL_MACHINE` hive, like this (the version numbers will of course vary between different versions of Apache):

```
HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software Foundation\Apache\2.2.2
```

Correspondingly, if you chose to install Apache for the current user only, the key is located under the `HKEY_CURRENT_USER` hive, the contents of which are dependent of the user currently logged on:

```
HKEY_CURRENT_USER\SOFTWARE\Apache Software Foundation\Apache\2.2.2
```

This key is compiled into the server and can enable you to test new versions without affecting the current version. Of course, you must take care not to install the new version in the same directory as another version.

If you did not do a binary install, Apache will in some scenarios complain about the missing registry key. This warning can be ignored if the server was otherwise able to find its configuration file.

The value of this key is the `ServerRoot` directory which contains the `conf` subdirectory. When Apache starts it reads the `httpd.conf` file from that directory. If this file contains a `ServerRoot` directive which contains a different directory from the one obtained from the registry key above, Apache will forget the registry key and use the directory from the configuration file. If you copy the Apache directory or configuration files to a new location it is vital that you update the `ServerRoot` directive in the

httpd.conf file to reflect the new location.



Testing the installation

After starting Apache (either in a console window or as a service) it will be listening on port 80 (unless you changed the `Listen` directive in the configuration files or installed Apache only for the current user). To connect to the server and access the default page, launch a browser and enter this URL:

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

Apache should respond with a welcome page and you should see "It Works!". If nothing happens or you get an error, look in the `error.log` file in the `logs` subdirectory. If your host is not connected to the net, or if you have serious problems with your DNS (Domain Name Service) configuration, you may have to use this URL:

```
http://127.0.0.1/
```

If you happen to be running Apache on an alternate port, you need to explicitly put that in the URL:

```
http://127.0.0.1:8080/
```

Once your basic installation is working, you should configure it properly by editing the files in the `conf` subdirectory. Again, if you change the configuration of the Windows NT service for Apache, first attempt to start it from the command line to make sure that the service starts with no errors.

Because Apache **cannot** share the same port with another TCP/IP application, you may need to stop, uninstall or reconfigure certain other services before running Apache. These conflicting services include other WWW servers, some firewall implementations, and even some client applications (such as Skype) which will use port 80 to attempt to bypass firewall issues.



Access to files over the network can be specified using two mechanisms provided by Windows:

Mapped drive letters

e.g., Alias /images/ Z:/

UNC paths

e.g., Alias /images/ //imagehost/www/images/

Mapped drive letters allow the administrator to maintain the mapping to a specific machine and path outside of the Apache httpd configuration. However, these mappings are associated only with interactive sessions and are not directly available to Apache httpd when it is started as a service. **Use only UNC paths for network resources in httpd.conf** so that the resources can be accessed consistently regardless of how Apache httpd is started. (Arcane and error prone procedures may work around the restriction on mapped drive letters, but this is not recommended.)

Example DocumentRoot with UNC path

```
DocumentRoot //dochost/www/html/
```

Example DocumentRoot with IP address in UNC path

```
DocumentRoot //192.168.1.50/docs/
```

Example Alias and corresponding Directory with UNC path

```
Alias /images/ //imagehost/www/images/  
  
<Directory //imagehost/www/images/>  
...  
<Directory>
```

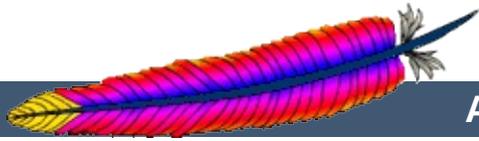
When running Apache httpd as a service, you must create a separate account in order to access network resources, as described above.



- If more than a few dozen piped loggers are used on an operating system instance, scaling up the "desktop heap" is often necessary. For more detailed information, refer to the [piped logging](#) documentation.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Platform Specific Notes](#)

Compiling Apache for Microsoft Windows

There are many important points before you begin compiling Apache. See [Using Apache with Microsoft Windows](#) before you begin.



Requirements

Compiling Apache requires the following environment to be properly installed:

- Disk Space

Make sure you have at least 200 MB of free disk space available. After installation Apache requires approximately 80 MB of disk space, plus space for log and cache files, which can grow rapidly. The actual disk space requirements will vary considerably based on your chosen configuration and any third-party modules or libraries, especially when OpenSSL is also built. Because many files are text and very easily compressed, NTFS filesystem compression cuts these requirements in half.

- Appropriate Patches

The httpd binary is built with the help of several patches to third party packages, which ensure the released code is buildable and debuggable. These patches are available and distributed from http://www.apache.org/dist/httpd/binaries/win32/patches_applied/ and are recommended to be applied to obtain identical results as the "official" ASF distributed binaries.

- Microsoft Visual C++ 6.0 (Visual Studio 97) or later.

Apache can be built using the command line tools, or from within the Visual Studio IDE Workbench. The command line build requires the environment to reflect the PATH, INCLUDE, LIB and other variables that can be configured with the `vcvars32.bat` script.

You may want the Visual Studio Processor Pack for your older version of Visual Studio, or a full (not Express) version of newer Visual Studio editions, for the ml.exe assembler. This will allow you to build OpenSSL, if desired, using the more efficient

assembly code implementation.

Only the Microsoft compiler tool chain is actively supported by the active httpd contributors. Although the project regularly accepts patches to ensure MinGW and other alternative builds work and improve upon them, they are not actively maintained and are often broken in the course of normal development.

- Updated Microsoft Windows Platform SDK, February 2003 or later.

An appropriate Windows Platform SDK is included by default in the full (not express/lite) versions of Visual C++ 7.1 (Visual Studio 2002) and later, these users can ignore these steps unless explicitly choosing a newer or different version of the Platform SDK.

To use Visual C++ 6.0 or 7.0 (Studio 2000 .NET), the Platform SDK environment must be prepared using the `setenv.bat` script (installed by the Platform SDK) before starting the command line build or launching the `msdev/devenv` GUI environment. Installing the Platform SDK for Visual Studio Express versions (2003 and later) should adjust the default environment appropriately.

```
"c:\Program Files\Microsoft Visual Studio\VC98\Bin\VCVARS32"  
"c:\Program Files\Platform SDK\setenv.bat"
```

- Perl and awk

Several steps recommended here require a perl interpreter during the build preparation process, but it is otherwise not required.

To install Apache within the build system, several files are modified using the `awk.exe` utility. `awk` was chosen since it is a very small download (compared with Perl or WSH/VB) and accomplishes the task of modifying configuration files upon installation. Brian Kernighan's <http://www.cs.princeton.edu/~bwk/btl.mirror/> site has a

compiled native Win32 binary,
<http://www.cs.princeton.edu/~bwk/btl.mirror/awk95.exe> which you must save with the name `awk.exe` (rather than `awk95.exe`).

If `awk.exe` is not found, `Makefile.win`'s `install` target will not perform substitutions in the installed `.conf` files. You must manually modify the installed `.conf` files to allow the server to start. Search and replace all `"@token@"` tags as appropriate.

The Visual Studio IDE will only find `awk.exe` from the `PATH`, or executable path specified in the menu option `Tools -> Options -> (Projects ->) Directories`. Ensure `awk.exe` is in your system path.

Also note that if you are using Cygwin tools (<http://www.cygwin.com/>) the `awk` utility is named `gawk.exe` and that the file `awk.exe` is really a symlink to the `gawk.exe` file. The Windows command shell does not recognize symlinks, and because of this building `InstallBin` will fail. A workaround is to delete `awk.exe` from the cygwin installation and copy `gawk.exe` to `awk.exe`. Also note the cygwin/mingw ports of `gawk 3.0.x` were buggy, please upgrade to `3.1.x` before attempting to use any `gawk` port.

- [Optional] `zlib` library (for `mod_deflate`)

`Zlib` must be installed into a `src\lib` subdirectory named `zlib`. This must be built in-place. `Zlib` can be obtained from <http://www.zlib.net/> - the `mod_deflate` is confirmed to work correctly with version `1.2.3`.

```
nmake -f win32\Makefile.msc  
nmake -f win32\Makefile.msc test
```

- [Optional] `OpenSSL` libraries (for `mod_ssl` and `ab.exe` with `ssl`)

support)

The OpenSSL library is cryptographic software. The country in which you currently reside may have restrictions on the import, possession, use, and/or re-export to another country, of encryption software. BEFORE using any encryption software, please check your country's laws, regulations and policies concerning the import, possession, or use, and re-export of encryption software, to see if this is permitted. See <http://www.wassenaar.org/> for more information.

Configuring and building OpenSSL requires perl to be installed.

OpenSSL must be installed into a `src\lib` subdirectory named `openssl`, obtained from <http://www.openssl.org/source/>, in order to compile `mod_ssl` or the `abs.exe` project, which is `ab.c` with SSL support enabled. To prepare OpenSSL to be linked to Apache `mod_ssl` or `abs.exe`, and disable patent encumbered features in OpenSSL, you might use the following build commands:

```
perl Configure no-rc5 no-idea enable-mdc2 enable-zlib VC-WIN32 -  
Ipath/to/src\lib\zlib -Lpath/to/src\lib\zlib  
ms\do_masm.bat  
nmake -f ms\ntdll.mak
```

It is not advisable to use `zlib-dynamic`, as that transfers the cost of deflating SSL streams to the first request which must load the `zlib.dll`. Note the suggested patch enables the `-L` flag to work with windows builds, corrects the name of `zdll.lib` and ensures `.pdb` files are generated for troubleshooting. If the assembler is not installed, you would add `no-asm` above and use `ms\do_ms.bat` instead of the `ms\do_masm.bat` script.

- [Optional] Database libraries (for `mod_dbd` and `mod_authn_dbm`)

The apr-util library exposes dbm (keyed database) and dbd (query oriented database) client functionality to the httpd server and its modules, such as authentication and authorization. The sdbm dbm and odbc dbd providers are compiled unconditionally.

The dbd support includes the Oracle instantclient package, MySQL, PostgreSQL and sqlite. To build these all, for example, set up the LIB to include the library path, INCLUDE to include the headers path, and PATH to include the dll bin path of all four SDK's, and set the DBD_LIST environment variable to inform the build which client driver SDKs are installed correctly, e.g.;

```
set DBD_LIST=sqlite3 pgsql oracle mysql
```

Similarly, the dbm support can be extended with DBM_LIST to build a Berkeley DB provider (db) and/or gdbm provider, by similarly configuring LIB, INCLUDE and PATH first to ensure the client library libs and headers are available.

```
set DBM_LIST=db gdbm
```

Depending on the choice of database distributions, it may be necessary to change the actual link target name (e.g. gdbm.lib vs. libgdb.lib) that are listed in the corresponding .dsp/.mak files within the directories srclib\apr-util\dbd or ...dbm.

See the README-win32.txt file for more hints on obtaining the various database driver SDKs.



`Makefile.win` is the top level Apache makefile. To compile Apache on Windows, simply use one of the following commands to build the release or debug flavor:

```
nmake /f Makefile.win _apacher  
nmake /f Makefile.win _apached
```

Either command will compile Apache. The latter will disable optimization of the resulting files, making it easier to single step the code to find bugs and track down problems.

You can add your apr-util dbd and dbm provider choices with the additional make (environment) variables `DBD_LIST` and `DBM_LIST`, see the comments about [Optional] Database libraries, above. Review the initial comments in `Makefile.win` for additional options that can be provided when invoking the build.



Apache can also be compiled using VC++'s Visual Studio development environment. To simplify this process, a Visual Studio workspace, Apache .dsw, is provided. This workspace exposes the entire list of working .dsp projects that are required for the complete Apache binary release. It includes dependencies between the projects to assure that they are built in the appropriate order.

Open the Apache .dsw workspace, and select InstallBin (Release or Debug build, as desired) as the Active Project. InstallBin causes all related project to be built, and then invokes Makefile.win to move the compiled executables and dlls. You may personalize the INSTDIR= choice by changing InstallBin's Settings, General tab, Build command line entry. INSTDIR defaults to the /Apache2 directory. If you only want a test compile (without installing) you may build the BuildBin project instead.

The .dsp project files are distributed in Visual Studio 6.0 (98) format. Visual C++ 5.0 (97) will recognize them. Visual Studio 2002 (.NET) and later users must convert Apache .dsw plus the .dsp files into an Apache .sln plus .msproj files. Be sure you reconvert the .msproj file again if its source .dsp file changes! This is really trivial, just open Apache .dsw in the VC++ 7.0 IDE once again and reconvert.

There is a flaw in the .vcproj conversion of .dsp files. devenv.exe will mis-parse the /D flag for RC flags containing long quoted /D'efines which contain spaces. The command:

```
perl srclib\apr\build\cvtdsp.pl -2005
```

will convert the /D flags for RC flags to use an alternate, parseable syntax; unfortunately this syntax isn't supported by Visual Studio 97 or its exported .mak files. These /D flags are used to pass the long description of the mod_apachemodule.so files to the shared .rc

resource version-identifier build.

Visual Studio 2002 (.NET) and later users should also use the Build menu, Configuration Manager dialog to uncheck both the Debug and Release Solution modules `abs`, `mod_deflate` and `mod_ssl` components, as well as every component starting with `apr_db*`. These modules are built by invoking `nmake`, or the IDE directly with the `BinBuild` target, which builds those modules conditionally if the `src/lib` directories `openssl` and/or `zlib` exist, and based on the setting of `DBD_LIST` and `DBM_LIST` environment variables.



Exported .mak files pose a greater hassle, but they are required for Visual C++ 5.0 users to build `mod_ssl`, `abs` (`ab` with SSL support) and/or `mod_deflate`. The .mak files also support a broader range of C++ tool chain distributions, such as Visual Studio Express.

You must first build all projects in order to create all dynamic auto-generated targets, so that dependencies can be parsed correctly. Build the entire project from within the Visual Studio 6.0 (98) IDE, using the `BuildAll` target, then use the Project Menu Export for all makefiles (checking on "with dependencies".) Run the following command to correct absolute paths into relative paths so they will build anywhere:

```
perl srclib\apr\build\fixwin32mak.pl
```

You must type this command from the *top level* directory of the httpd source tree. Every .mak and .dep project file within the current directory and below will be corrected, and the timestamps adjusted to reflect the .dsp.

Always review the generated .mak and .dep files for Platform SDK or other local, machine specific file paths. The `DevStudio\Common\MSDev98\bin\ (VC6)` directory contains a `sysincl.dat` file, which lists all exceptions. Update this file (including both forward and backslashed paths, such as both `sys/time.h` and `sys\time.h`) to ignore such newer dependencies. Including local-install paths in a distributed .mak file will cause the build to fail completely.

If you contribute back a patch that revises project files, we must commit project files in Visual Studio 6.0 format. Changes should be simple, with minimal compilation and linkage flags that can be recognized by all Visual Studio environments.



Once Apache has been compiled, it needs to be installed in its server root directory. The default is the \Apache2 directory, of the same drive.

To build and install all the files into the desired folder *dir* automatically, use one of the following nmake commands:

```
nmake /f Makefile.win installr INSTDIR=dir  
nmake /f Makefile.win installd INSTDIR=dir
```

The *dir* argument to INSTDIR provides the installation directory; it can be omitted if Apache is to be installed into \Apache22 (of the current drive).

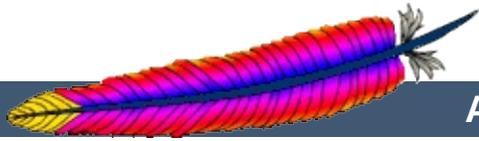


Warning about building Apache from the development tree

Note only the .dsp files are maintained between release builds. The .mak files are NOT regenerated, due to the tremendous waste of reviewer's time. Therefore, you cannot rely on the NMAKE commands above to build revised .dsp project files unless you then export all .mak files yourself from the project. This is unnecessary if you build from within the Microsoft Developer Studio environment.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Platform Specific Notes](#)

Using Apache With Novell NetWare

This document explains how to install, configure and run Apache 2.0 under Novell NetWare 6.0 and above. If you find any bugs, or wish to contribute in other ways, please use our [bug reporting page](#).

The bug reporting page and dev-httpd mailing list are *not* provided to answer questions about configuration or running Apache. Before you submit a bug report or request, first consult this document, the [Frequently Asked Questions](#) page and the other relevant documentation topics. If you still have a question or problem, post it to the [novell.devsup.webserver](#) newsgroup, where many Apache users are more than willing to answer new and obscure questions about using Apache on NetWare.

Most of this document assumes that you are installing Apache from a binary distribution. If you want to compile Apache yourself (possibly to help with development, or to track down bugs), see the section on [Compiling Apache for NetWare](#) below.



Requirements

Apache 2.0 is designed to run on NetWare 6.0 service pack 3 and above. If you are running a service pack less than SP3, you must install the latest [NetWare Libraries for C \(LibC\)](#).

NetWare service packs are available [here](#).

Apache 2.0 for NetWare can also be run in a NetWare 5.1 environment as long as the latest service pack or the latest version of the [NetWare Libraries for C \(LibC\)](#) has been installed . **WARNING:** Apache 2.0 for NetWare has not been targeted for or tested in this environment.



Information on the latest version of Apache can be found on the Apache web server at <http://www.apache.org/>. This will list the current release, any more recent alpha or beta-test releases, together with details of mirror web and anonymous ftp sites. Binary builds of the latest releases of Apache 2.0 for NetWare can be downloaded from [here](#).



Installing Apache for NetWare

There is no Apache install program for NetWare currently. If you are building Apache 2.0 for NetWare from source, you will need to copy the files over to the server manually.

Follow these steps to install Apache on NetWare from the binary download (assuming you will install to `sys:/apache2`):

- Unzip the binary download file to the root of the SYS: volume (may be installed to any volume)
- Edit the `httpd.conf` file setting [ServerRoot](#) and [ServerName](#) along with any file path values to reflect your correct server settings
- Add `SYS:/APACHE2` to the search path, for example:

```
SEARCH ADD SYS:\APACHE2
```

Follow these steps to install Apache on NetWare manually from your own build source (assuming you will install to `sys:/apache2`):

- Create a directory called Apache2 on a NetWare volume
- Copy `APACHE2.NLM`, `APRLIB.NLM` to `SYS:/APACHE2`
- Create a directory under `SYS:/APACHE2` called BIN
- Copy `HTDIGEST.NLM`, `HTPASSWORD.NLM`, `HTDBM.NLM`, `LOGRES.NLM`, `ROTLOGS.NLM` to `SYS:/APACHE2/BIN`
- Create a directory under `SYS:/APACHE2` called CONF
- Copy the `HTTPD-STD.CONF` file to the `SYS:/APACHE2/CONF` directory and rename to `HTTPD.CONF`
- Copy the `MIME.TYPES`, `CHARSET.CONV` and `MAGIC` files to `SYS:/APACHE2/CONF` directory
- Copy all files and subdirectories in `\HTTPD-2.0\DOCS\ICONS` to `SYS:/APACHE2/ICONS`
- Copy all files and subdirectories in `\HTTPD-2.0\DOCS\MANUAL` to `SYS:/APACHE2/MANUAL`

- Copy all files and subdirectories in \HTTPD-2.0\DOCS\ERROR to SYS:/APACHE2/ERROR
- Copy all files and subdirectories in \HTTPD-2.0\DOCS\DOCR00T to SYS:/APACHE2/HTDOCS
- Create the directory SYS:/APACHE2/LOGS on the server
- Create the directory SYS:/APACHE2/CGI-BIN on the server
- Create the directory SYS:/APACHE2/MODULES and copy all nlm modules into the modules directory
- Edit the HTTPD.CONF file searching for all @@Value@@ markers and replacing them with the appropriate setting
- Add SYS:/APACHE2 to the search path, for example:

```
SEARCH ADD SYS:\APACHE2
```

Apache may be installed to other volumes besides the default SYS volume.

During the build process, adding the keyword "install" to the makefile command line will automatically produce a complete distribution package under the subdirectory DIST. Install Apache by simply copying the distribution that was produced by the makfiles to the root of a NetWare volume (see: [Compiling Apache for NetWare](#) below).



Running Apache on NetWare

To start Apache just type `apache` at the console. This will load apache in the OS address space. If you prefer to load Apache in a protected address space you may specify the address space with the `load` statement as follows:

```
load address space = apache2 apache2
```

This will load Apache into an address space called `apache2`. Running multiple instances of Apache concurrently on NetWare is possible by loading each instance into its own protected address space.

After starting Apache, it will be listening to port 80 (unless you changed the `Listen` directive in the configuration files). To connect to the server and access the default page, launch a browser and enter the server's name or address. This should respond with a welcome page, and a link to the Apache manual. If nothing happens or you get an error, look in the `error_log` file in the `logs` directory.

Once your basic installation is working, you should configure it properly by editing the files in the `conf` directory.

To unload Apache running in the OS address space just type the following at the console:

```
unload apache2
```

or

```
apache2 shutdown
```

If apache is running in a protected address space specify the address space in the `unload` statement:

```
unload address space = apache2 apache2
```

When working with Apache it is important to know how it will find the configuration files. You can specify a configuration file on the command line in two ways:

- -f specifies a path to a particular configuration file

```
apache2 -f "vol1:/my server/conf/my.conf"
```

```
apache -f test/test.conf
```

In these cases, the proper [ServerRoot](#) should be set in the configuration file.

If you don't specify a configuration file name with -f, Apache will use the file name compiled into the server, usually `conf/httpd.conf`. Invoking Apache with the -V switch will display this value labeled as `SERVER_CONFIG_FILE`. Apache will then determine its [ServerRoot](#) by trying the following, in this order:

- A `ServerRoot` directive via a -C switch.
- The -d switch on the command line.
- Current working directory
- The server root compiled into the server.

The server root compiled into the server is usually `sys:/apache2`. Invoking `apache` with the -V switch will display this value labeled as `HTTPD_ROOT`.

Apache 2.0 for NetWare includes a set of command line directives that can be used to modify or display information about the running instance of the web server. These directives are only available while Apache is running. Each of these directives must be preceded by the keyword `APACHE2`.

RESTART

Instructs Apache to terminate all running worker threads as they become idle, reread the configuration file and restart each worker thread based on the new configuration.

VERSION

Displays version information about the currently running instance of Apache.

MODULES

Displays a list of loaded modules both built-in and external.

DIRECTIVES

Displays a list of all available directives.

SETTINGS

Enables or disables the thread status display on the console. When enabled, the state of each running threads is displayed on the Apache console screen.

SHUTDOWN

Terminates the running instance of the Apache web server.

HELP

Describes each of the runtime directives.

By default these directives are issued against the instance of Apache running in the OS address space. To issue a directive against a specific instance running in a protected address space, include the -p parameter along with the name of the address space. For more information type "apache2 Help" on the command line.



Configuring Apache for NetWare

Apache is configured by reading configuration files usually stored in the conf directory. These are the same as files used to configure the Unix version, but there are a few different directives for Apache on NetWare. See the [Apache module documentation](#) for all the available directives.

The main differences in Apache for NetWare are:

- Because Apache for NetWare is multithreaded, it does not use a separate process for each request, as Apache does on some Unix implementations. Instead there are only threads running: a parent thread, and multiple child or worker threads which handle the requests.

Therefore the "process"-management directives are different:

MaxRequestsPerChild - Like the Unix directive, this controls how many requests a worker thread will serve before exiting. The recommended default, `MaxRequestsPerChild 0`, causes the thread to continue servicing request indefinitely. It is recommended on NetWare, unless there is some specific reason, that this directive always remain set to 0.

StartThreads - This directive tells the server how many threads it should start initially. The recommended default is `StartThreads 50`.

MinSpareThreads - This directive instructs the server to spawn additional worker threads if the number of idle threads ever falls below this value. The recommended default is `MinSpareThreads 10`.

MaxSpareThreads - This directive instructs the server to begin terminating worker threads if the number of idle threads ever exceeds this value. The recommended default is

MaxSpareThreads 100.

[MaxThreads](#) - This directive limits the total number of work threads to a maximum value. The recommended default is ThreadsPerChild 250.

[ThreadStackSize](#) - This directive tells the server what size of stack to use for the individual worker thread. The recommended default is ThreadStackSize 65536.

- The directives that accept filenames as arguments must use NetWare filenames instead of Unix names. However, because Apache uses Unix-style names internally, forward slashes must be used rather than backslashes. It is recommended that all rooted file paths begin with a volume name. If omitted, Apache will assume the SYS: volume which may not be correct.
- Apache for NetWare has the ability to load modules at runtime, without recompiling the server. If Apache is compiled normally, it will install a number of optional modules in the \Apache2\modules directory. To activate these, or other modules, the [LoadModule](#) directive must be used. For example, to activate the status module, use the following:

```
LoadModule status_module modules/status.nlm
```

Information on [creating loadable modules](#) is also available.

Additional NetWare specific directives:

- [CGIMapExtension](#) - This directive maps a CGI file extension to a script interpreter.
- [SecureListen](#) - Enables SSL encryption for a specified port.

- [NWSSLTrustedCerts](#) - Adds trusted certificates that are used to create secure connections to proxied servers.
- [NWSSLUpgradeable](#) - Allow a connection created on the specified address/port to be upgraded to an SSL connection.



Compiling Apache for NetWare

Compiling Apache requires MetroWerks CodeWarrior 6.x or higher. Once Apache has been built, it can be installed to the root of any NetWare volume. The default is the `sys:/Apache2` directory.

Before running the server you must fill out the `conf` directory. Copy the file `HTTPD-STD.CONF` from the distribution `conf` directory and rename it to `HTTPD.CONF`. Edit the `HTTPD.CONF` file searching for all `@@Value@@` markers and replacing them with the appropriate setting. Copy over the `conf/magic` and `conf/mime.types` files as well. Alternatively, a complete distribution can be built by including the keyword `install` when invoking the makefiles.

Requirements:

The following development tools are required to build Apache 2.0 for NetWare:

- Metrowerks CodeWarrior 6.0 or higher with the [NetWare PDK 3.0](#) or higher.
- [NetWare Libraries for C \(LibC\)](#)
- [LDAP Libraries for C](#)
- [ZLIB Compression Library source code](#)
- AWK utility (`awk`, `gawk` or similar). AWK can be downloaded from <http://developer.novell.com/ndk/apache.htm>. The utility must be found in your windows path and must be named `awk.exe`.
- To build using the makefiles, you will need GNU make version 3.78.1 (GMake) available at <http://developer.novell.com/ndk/apache.htm>.

Building Apache using the NetWare makefiles:

- Set the environment variable `NOVELLIBC` to the location of the NetWare Libraries for C SDK, for example:

```
Set NOVELLIBC=c:\novell\ndk\libc
```

- Set the environment variable METROWERKS to the location where you installed the Metrowerks CodeWarrior compiler, for example:

```
Set METROWERKS=C:\Program Files\Metrowerks\CodeWarrior
```

If you installed to the default location C:\Program Files\Metrowerks\CodeWarrior, you don't need to set this.

- Set the environment variable LDAPSDK to the location where you installed the LDAP Libraries for C, for example:

```
Set LDAPSDK=c:\Novell\NDK\cldapsdk\NetWare\libc
```

- Set the environment variable ZLIBSDK to the location where you installed the source code for the ZLib Library, for example:

```
Set ZLIBSDK=D:\NOVELL\zlib
```

- Set the environment variable AP_WORK to the full path of the httpd source code directory.

```
Set AP_WORK=D:\httpd-2.0.x
```

- Set the environment variable APR_WORK to the full path of the apr source code directory. Typically \httpd\src\lib\apr but the APR project can be outside of the httpd directory structure.

```
Set APR_WORK=D:\apr-1.x.x
```

- Set the environment variable APU_WORK to the full path of the apr-util source code directory. Typically \httpd\src\lib\apr-util but the APR-UTIL project can be outside of the httpd directory structure.

```
Set APU_WORK=D:\apr-util-1.x.x
```

- Make sure that the path to the AWK utility and the GNU make utility (gmake.exe) have been included in the system's PATH environment variable.
- Download the source code and unzip to an appropriate directory on your workstation.
- Change directory to `\httpd-2.0` and build the prebuild utilities by running `"gmake -f nwgnumakefile prebuild"`. This target will create the directory `\httpd-2.0\nwprebuild` and copy each of the utilities to this location that are necessary to complete the following build steps.
- Copy the files `\httpd-2.0\nwprebuild\GENCHARS.nlm` and `\httpd-2.0\nwprebuild\DFTABLES.nlm` to the SYS: volume of a NetWare server and run them using the following commands:

```
SYS:\genchars > sys:\test_char.h  
SYS:\dftables sys:\chartables.c
```

- Copy the files `test_char.h` and `chartables.c` to the directory `\httpd-2.0\os\netware` on the build machine.
- Change directory to `\httpd-2.0` and build Apache by running `"gmake -f nwgnumakefile"`. You can create a distribution directory by adding an install parameter to the command, for example:

```
gmake -f nwgnumakefile install
```

Additional make options

- `gmake -f nwgnumakefile`
Builds release versions of all of the binaries and copies them to a `\release` destination directory.
- `gmake -f nwgnumakefile DEBUG=1`

Builds debug versions of all of the binaries and copies them to a `\debug` destination directory.

- `gmake -f nwgnumakefile install`
Creates a complete Apache distribution with binaries, docs and additional support files in a `\dist\Apache2` directory.
- `gmake -f nwgnumakefile prebuild`
Builds all of the prebuild utilities and copies them to the `\nwprebuild` directory.
- `gmake -f nwgnumakefile installdev`
Same as `install` but also creates a `\lib` and `\include` directory in the destination directory and copies headers and import files.
- `gmake -f nwgnumakefile clean`
Cleans all object files and binaries from the `\release.o` or `\debug.o` build areas depending on whether `DEBUG` has been defined.
- `gmake -f nwgnumakefile clobber_all`
Same as `clean` and also deletes the distribution directory if it exists.

Additional environment variable options

- To build all of the experimental modules, set the environment variable `EXPERIMENTAL`:

```
Set EXPERIMENTAL=1
```

- To build Apache using standard BSD style sockets rather than Winsock, set the environment variable `USE_STDSOCKETS`:

```
Set USE_STDSOCKETS=1
```

Building mod_ssl for the NetWare platform

By default Apache for NetWare uses the built-in module `mod_nw_ssl` to provide SSL services. This module simply enables the native SSL services implemented in NetWare OS to handle all encryption for a given port. Alternatively, `mod_ssl` can also be used in the same manner as on other platforms.

Before `mod_ssl` can be built for the NetWare platform, the OpenSSL libraries must be provided. This can be done through the following steps:

- Download the recent OpenSSL 0.9.8 release source code from the [OpenSSL Source](#) page (older 0.9.7 versions need to be patched and are therefore not recommended).
- Edit the file `NetWare/set_env.bat` and modify any tools and utilities paths so that they correspond to your build environment.
- From the root of the OpenSSL source directory, run the following scripts:

```
Netware\set_env netware-libc
Netware\build netware-libc
```

For performance reasons you should enable to build with ASM code. Download NASM from the [SF site](#). Then configure OpenSSL to use ASM code:

```
Netware\build netware-libc nw-nasm enable-mdc2 enable-md5
```

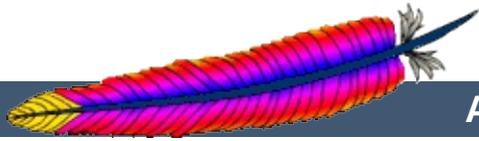
Warning: dont use the CodeWarrior Assembler - it produces broken code!

- Before building Apache, set the environment variable `OSSLSDK` to the full path to the root of the openssl source code directory, and set `WITH_MOD_SSL` to 1.

```
Set OSSLSDK=d:\openssl-0.9.8x
Set WITH_MOD_SSL=1
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Platform Specific Notes](#)

Running a High-Performance Web Server on HPUX

Date: Wed, 05 Nov 1997 16:59:34 -0800
From: Rick Jones <raj@cup.hp.com>
Reply-To: raj@cup.hp.com
Organization: Network Performance
Subject: HP-UX tuning tips

Here are some tuning tips for HP-UX to add to the tuning page.

For HP-UX 9.X: Upgrade to 10.20

For HP-UX 10.[00|01|10]: Upgrade to 10.20

For HP-UX 10.20:

Install the latest cumulative ARPA Transport Patch. This will allow you to configure the size of the TCP connection lookup hash table. The default is 256 buckets and must be set to a power of two. This is accomplished with adb against the *disc* image of the kernel. The variable name is tcp_hash_size. Notice that it's critically important that you use "w" to write a 32 bit quantity, not "W" to write a 16 bit value when patching the disc image because the tcp_hash_size variable is a 32 bit quantity.

How to pick the value? Examine the output of <ftp://ftp.cup.hp.com/dist/networking/tools/connhist> and see how many total TCP connections exist on the system. You probably want that number divided by the hash table size to be reasonably small, say less than 10. Folks can look at HP's SPECweb96 disclosures for some common settings. These can be found at <http://www.specbench.org/>. If an HP-UX system was performing at 1000 SPECweb96 connections per second, the TIME_WAIT time of 60 seconds would mean 60,000 TCP "connections" being tracked.

Folks can check their listen queue depths with <ftp://ftp.cup.hp.com/dist/networking/misc/listenq>.

If folks are running Apache on a PA-8000 based system, they should consider "chat'ring" the Apache executable to have a large page size. This would be "chatr +pi L <BINARY>". The GID of the running executable must have MLOCK privileges. Setprivgrp(1m) should be consulted for assigning MLOCK. The change can be validated by running Glance and examining the memory regions of the server(s) to make sure that they show a non-trivial fraction of the text segment being locked.

If folks are running Apache on MP systems, they might consider writing a small program that uses mpctl() to bind processes to processors. A simple pid % numcpu algorithm is probably sufficient. This might even go into the source code.

If folks are concerned about the number of FIN_WAIT_2 connections, they can use nettune to shrink the value of tcp_keepstart. However, they should be careful there - certainly do not make it less than oh two to four minutes. If tcp_hash_size has been set well, it is probably OK to let the FIN_WAIT_2's take longer to timeout (perhaps even the default two hours) - they will not on average have a big impact on performance.

There are other things that could go into the code base, but that might be left for another email. Feel free to drop me a message if you or others are interested.

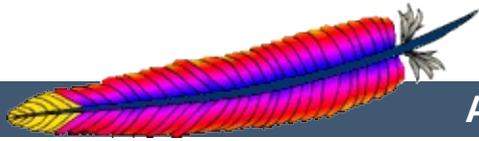
sincerely,

rick jones

<http://www.netperf.org/netperf/>

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Platform Specific Notes](#)

The Apache EBCDIC Port

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.



Version 1.3 of the Apache HTTP Server is the first version which includes a port to a (non-ASCII) mainframe machine which uses the EBCDIC character set as its native codeset.

(It is the SIEMENS family of mainframes running the [BS2000/OSD operating system](#). This mainframe OS nowadays features a SVR4-derived POSIX subsystem).

The port was started initially to

- prove the feasibility of porting [the Apache HTTP server](#) to this platform
- find a "worthy and capable" successor for the venerable [CERN-3.0](#) daemon (which was ported a couple of years ago), and to
- prove that Apache's preforking process model can on this platform easily outperform the accept-fork-serve model used by CERN by a factor of 5 or more.

This document serves as a rationale to describe some of the design decisions of the port to this machine.



One objective of the EBCDIC port was to maintain enough backwards compatibility with the (EBCDIC) CERN server to make the transition to the new server attractive and easy. This required the addition of a configurable method to define whether a HTML document was stored in ASCII (the only format accepted by the old server) or in EBCDIC (the native document format in the POSIX subsystem, and therefore the only realistic format in which the other POSIX tools like grep or sed could operate on the documents). The current solution to this is a "pseudo-MIME-format" which is intercepted and interpreted by the Apache server (see below). Future versions might solve the problem by defining an "ebcdic-handler" for all documents which must be converted.



Since all Apache input and output is based upon the BUFF data type and its methods, the easiest solution was to add the conversion to the BUFF handling routines. The conversion must be settable at any time, so a BUFF flag was added which defines whether a BUFF object has currently enabled conversion or not. This flag is modified at several points in the HTTP protocol:

- **set** before a request is received (because the request and the request header lines are always in ASCII format)
- **set/unset** when the request body is received - depending on the content type of the request body (because the request body may contain ASCII text or a binary file)
- **set** before a reply header is sent (because the response header lines are always in ASCII format)
- **set/unset** when the response body is sent - depending on the content type of the response body (because the response body may contain text or a binary file)



1. The relevant changes in the source are `#ifdef`'ed into two categories:

`#ifdef CHARSET_EBCDIC`

Code which is needed for any EBCDIC based machine. This includes character translations, differences in contiguity of the two character sets, flags which indicate which part of the HTTP protocol has to be converted and which part doesn't *etc.*

`#ifdef _OSD_POSIX`

Code which is needed for the SIEMENS BS2000/OSD mainframe platform only. This deals with include file differences and socket implementation topics which are only required on the BS2000/OSD platform.

2. The possibility to translate between ASCII and EBCDIC at the socket level (on BS2000 POSIX, there is a socket option which supports this) was intentionally *not* chosen, because the byte stream at the HTTP protocol level consists of a mixture of protocol related strings and non-protocol related raw file data. HTTP protocol strings are always encoded in ASCII (the GET request, any Header: lines, the chunking information *etc.*) whereas the file transfer parts (*i.e.*, GIF images, CGI output *etc.*) should usually be just "passed through" by the server. This separation between "protocol string" and "raw data" is reflected in the server code by functions like `bgets()` or `rvputs()` for strings, and functions like `bwrite()` for binary data. A global translation of everything would therefore be inadequate.

(In the case of text files of course, provisions must be made so that EBCDIC documents are always served in ASCII)

3. This port therefore features a built-in protocol level conversion for the server-internal strings (which the compiler translated to EBCDIC strings) and thus for all server-generated documents. The hard

coded ASCII escapes `\012` and `\015` which are ubiquitous in the server code are an exception: they are already the binary encoding of the ASCII `\n` and `\r` and must not be converted to ASCII a second time. This exception is only relevant for server-generated strings; and *external* EBCDIC documents are not expected to contain ASCII newline characters.

4. By examining the call hierarchy for the BUFF management routines, I added an "ebcdic/ascii conversion layer" which would be crossed on every `puts/write/get/gets`, and a conversion flag which allowed enabling/disabling the conversions on-the-fly. Usually, a document crosses this layer twice from its origin source (a file or CGI output) to its destination (the requesting client): `file -> Apache`, and `Apache -> client`.

The server can now read the header lines of a CGI-script output in EBCDIC format, and then find out that the remainder of the script's output is in ASCII (like in the case of the output of a WWW Counter program: the document body contains a GIF image). All header processing is done in the native EBCDIC format; the server then determines, based on the type of document being served, whether the document body (except for the chunking information, of course) is in ASCII already or must be converted from EBCDIC.

5. For Text documents (MIME types `text/plain`, `text/html` etc.), an implicit translation to ASCII can be used, or (if the users prefer to store some documents in raw ASCII form for faster serving, or because the files reside on a NFS-mounted directory tree) can be served without conversion.

Example:

to serve files with the suffix `.ahtml` as a raw ASCII `text/html` document without implicit conversion (and suffix `.ascii` as ASCII `text/plain`), use the directives:

```
AddType text/x-ascii-html .html  
AddType text/x-ascii-plain .ascii
```

Similarly, any `text/foo` MIME type can be served as "raw ASCII" by configuring a MIME type "`text/x-ascii-foo`" for it using `AddType`.

6. Non-text documents are always served "binary" without conversion. This seems to be the most sensible choice for, *e.g.*, GIF/ZIP/AU file types. This of course requires the user to copy them to the mainframe host using the "`rcp -b`" binary switch.
7. Server parsed files are always assumed to be in native (*i.e.*, EBCDIC) format as used on the machine, and are converted after processing.
8. For CGI output, the CGI script determines whether a conversion is needed or not: by setting the appropriate Content-Type, text files can be converted, or GIF output can be passed through unmodified. An example for the latter case is the `wwwcount` program which we ported as well.



Binary Files

All files with a Content - Type : which does not start with text / are regarded as *binary files* by the server and are not subject to any conversion. Examples for binary files are GIF images, gzip-compressed files and the like.

When exchanging binary files between the mainframe host and a Unix machine or Windows PC, be sure to use the ftp "binary" (TYPE I) command, or use the rcp -b command from the mainframe host (the -b switch is not supported in unix rcp's).

Text Documents

The default assumption of the server is that Text Files (*i.e.*, all files whose Content - Type : starts with text /) are stored in the native character set of the host, EBCDIC.

Server Side Included Documents

SSI documents must currently be stored in EBCDIC only. No provision is made to convert it from ASCII before processing.



Module	Status	Notes
<u>core</u>	+	
mod_access	+	
<u>mod_actions</u>	+	
<u>mod_alias</u>	+	
<u>mod_asis</u>	+	
mod_auth	+	
mod_auth_anon	+	
mod_auth_dbm	?	with own libdb.a
<u>mod_autoindex</u>	+	
<u>mod_cern_meta</u>	?	
<u>mod_cgi</u>	+	
mod_digest	+	
<u>mod_dir</u>	+	
<u>mod_so</u>	-	no shared libs
<u>mod_env</u>	+	
<u>mod_example</u>	-	(test bed only)
<u>mod_expires</u>	+	
<u>mod_headers</u>	+	
<u>mod_imagemap</u>	+	
<u>mod_include</u>	+	
<u>mod_info</u>	+	
mod_log_agent	+	
mod_log_config	+	
mod_log_referer	+	
<u>mod_mime</u>	+	
<u>mod_mime_magic</u>	?	not ported yet
<u>mod_negotiation</u>	+	

<u>mod_proxy</u>	+	
<u>mod_rewrite</u>	+	untested
<u>mod_setenvif</u>	+	
<u>mod_speling</u>	+	
<u>mod_status</u>	+	
<u>mod_unique_id</u>	+	
<u>mod_userdir</u>	+	
<u>mod_usertrack</u>	?	untested

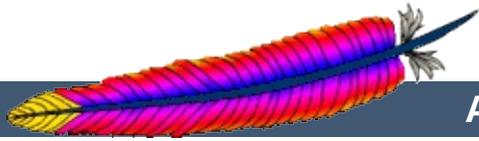


Third Party Modules Status

Module	Status	Notes
JK (Formerly mod_jserv)	-	JAVA still being ported.
mod_php3	+	mod_php3 runs fine, with LDAP and GD and FreeType libraries.
mod_put	?	untested
mod_session	-	untested

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

httpd - Apache Hypertext Transfer Protocol Server

httpd is the Apache HyperText Transfer Protocol (HTTP) server program. It is designed to be run as a standalone daemon process. When used like this it will create a pool of child processes or threads to handle requests.

In general, httpd should not be invoked directly, but rather should be invoked via [apachectl](#) on Unix-based systems or [as a service on Windows NT, 2000 and XP](#) and [as a console application on Windows 9x and ME](#).

See also

[Starting Apache httpd](#)

[Stopping Apache httpd](#)

[Configuration Files](#)

[Platform-specific Documentation](#)

[apachectl](#)



Synopsis

```
httpd [ -d serverroot ] [ -f config ] [ -C directive ]  
[ -c directive ] [ -D parameter ] [ -e level ] [ -E  
file ] [ -k start|restart|graceful|stop|graceful-stop  
] [ -R directory ] [ -h ] [ -l ] [ -L ] [ -S ] [ -t ]  
[ -v ] [ -V ] [ -X ] [ -M ] [ -T ]
```

On [Windows systems](#), the following additional arguments are available:

```
httpd [ -k install|config|uninstall ] [ -n name ] [ -w  
]
```



-d *serverroot*

Set the initial value for the [ServerRoot](#) directive to *serverroot*. This can be overridden by the `ServerRoot` directive in the configuration file. The default is `/usr/local/apache2`.

-f *config*

Uses the directives in the file *config* on startup. If *config* does not begin with a `/`, then it is taken to be a path relative to the [ServerRoot](#). The default is `conf/httpd.conf`.

-k *start | restart | graceful | stop | graceful-stop*

Signals `httpd` to start, restart, or stop. See [Stopping Apache httpd](#) for more information.

-C *directive*

Process the configuration *directive* before reading config files.

-c *directive*

Process the configuration *directive* after reading config files.

-D *parameter*

Sets a configuration *parameter* which can be used with [<IfDefine>](#) sections in the configuration files to conditionally skip or process commands at server startup and restart. Also can be used to set certain less-common startup parameters including `-DNO_DETACH` (prevent the parent from forking) and `-DFOREGROUND` (prevent the parent from calling `setsid()` et al).

-e *level*

Sets the [LogLevel](#) to *level* during server startup. This is useful for temporarily increasing the verbosity of the error messages to find problems during startup.

-E *file*

Send error messages during server startup to *file*.

-R *directory*

When the server is compiled using the SHARED_CORE rule, this specifies the *directory* for the shared object files.

-h

Output a short summary of available command line options.

-l

Output a list of modules compiled into the server. This will **not** list dynamically loaded modules included using the [LoadModule](#) directive.

-L

Output a list of directives together with expected arguments and places where the directive is valid.

-M

Dump a list of loaded Static and Shared Modules.

-S

Show the settings as parsed from the config file (currently only shows the virtualhost settings).

-T (Available in 2.2.17 and later)

Skip document root check at startup/restart.

-t

Run syntax tests for configuration files only. The program immediately exits after these syntax parsing tests with either a return code of 0 (Syntax OK) or return code not equal to 0 (Syntax Error). If `-D DUMP_VHOSTS` is also set, details of the virtual host configuration will be printed. If `-D DUMP_MODULES` is set, all loaded modules will be printed.

-v

Print the version of `ht tpd`, and then exit.

-V

Print the version and build parameters of `ht tpd`, and then exit.

-X

Run httpd in debug mode. Only one worker will be started and the server will not detach from the console.

The following arguments are available only on the [Windows platform](#):

-k `install|config|uninstall`

Install Apache httpd as a Windows NT service; change startup options for the Apache httpd service; and uninstall the Apache httpd service.

-n *name*

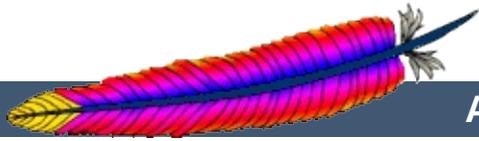
The *name* of the Apache httpd service to signal.

-w

Keep the console window open on error so that the error message can be read.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

ab - Apache HTTP server benchmarking tool

ab is a tool for benchmarking your Apache Hypertext Transfer Protocol (HTTP) server. It is designed to give you an impression of how your current Apache installation performs. This especially shows you how many requests per second your Apache installation is capable of serving.

See also

[httpd](#)



Synopsis

```
ab [ -A auth-username:password ] [ -b window-size ] [ -c concurrency ] [ -C cookie-name=value ] [ -d ] [ -e csv-file ] [ -f protocol ] [ -g gnuplot-file ] [ -h ] [ -H custom-header ] [ -i ] [ -k ] [ -n requests ] [ -p POST-file ] [ -P proxy-auth-username:password ] [ -q ] [ -r ] [ -s ] [ -S ] [ -t timelimit ] [ -T content-type ] [ -u PUT-file ] [ -v verbosity ] [ -V ] [ -w ] [ -x <table>-attributes ] [ -X proxy[:port] ] [ -y <tr>-attributes ] [ -z <td>-attributes ] [ -Z ciphersuite ] [http[s]://]hostname[:port]/path
```



-A *auth-username:password*

Supply BASIC Authentication credentials to the server. The username and password are separated by a single : and sent on the wire base64 encoded. The string is sent regardless of whether the server needs it (*i.e.*, has sent an 401 authentication needed).

-b *window-size*

Size of TCP send/receive buffer, in bytes.

-c *concurrency*

Number of multiple requests to perform at a time. Default is one request at a time.

-C *cookie-name=value*

Add a Cookie: line to the request. The argument is typically in the form of a *name=value* pair. This field is repeatable.

-d

Do not display the "percentage served within XX [ms] table". (legacy support).

-e *csv-file*

Write a Comma separated value (CSV) file which contains for each percentage (from 1% to 100%) the time (in milliseconds) it took to serve that percentage of the requests. This is usually more useful than the 'gnuplot' file; as the results are already 'binned'.

-f *protocol*

Specify SSL/TLS protocol (SSL2, SSL3, TLS1, or ALL).

-g *gnuplot-file*

Write all measured values out as a 'gnuplot' or TSV (Tab separate values) file. This file can easily be imported into packages like Gnuplot, IDL, Mathematica, Igor or even Excel. The labels are on the first line of the file.

-h

Display usage information.

-H *custom-header*

Append extra headers to the request. The argument is typically in the form of a valid header line, containing a colon-separated field-value pair (*i.e.*, "Accept-Encoding: zip/zop;8bit").

-i

Do HEAD requests instead of GET.

-k

Enable the HTTP KeepAlive feature, *i.e.*, perform multiple requests within one HTTP session. Default is no KeepAlive.

-n *requests*

Number of requests to perform for the benchmarking session. The default is to just perform a single request which usually leads to non-representative benchmarking results.

-p *POST-file*

File containing data to POST. Remember to also set -T.

-P *proxy-auth-username:password*

Supply BASIC Authentication credentials to a proxy en-route. The username and password are separated by a single : and sent on the wire base64 encoded. The string is sent regardless of whether the proxy needs it (*i.e.*, has sent an 407 proxy authentication needed).

-q

When processing more than 150 requests, ab outputs a progress count on `stderr` every 10% or 100 requests or so. The -q flag will suppress these messages.

-r

Don't exit on socket receive errors.

-s

When compiled in (ab -h will show you) use the SSL protected `https` rather than the `http` protocol. This feature is experimental and very rudimentary. You probably do not want to use it.

-S

Do not display the median and standard deviation values, nor display the warning/error messages when the average and median are more than one or two times the standard deviation apart. And default to the min/avg/max values. (legacy support).

-t *timelimit*

Maximum number of seconds to spend for benchmarking. This implies a `-n 50000` internally. Use this to benchmark the server within a fixed total amount of time. Per default there is no timelimit.

-T *content-type*

Content-type header to use for POST/PUT data, eg. `application/x-www-form-urlencoded`. Default: `text/plain`.

-u *PUT-file*

File containing data to PUT. Remember to also set `-T`.

-v *verbosity*

Set verbosity level - 4 and above prints information on headers, 3 and above prints response codes (404, 200, etc.), 2 and above prints warnings and info.

-V

Display version number and exit.

-w

Print out results in HTML tables. Default table is two columns wide, with a white background.

-x *<table>-attributes*

String to use as attributes for `<table>`. Attributes are inserted `<table here >`.

-X *proxy[:port]*

Use a proxy server for the requests.

-y *<tr>-attributes*

String to use as attributes for `<tr>`.

-z <td>-attributes

String to use as attributes for <td>.

-Z ciphersuite

Specify SSL/TLS cipher suite (See openssl ciphers).



The following list describes the values returned by ab:

Server Software

The value, if any, returned in the *server* HTTP header of the first successful response. This includes all characters in the header from beginning to the point a character with decimal value of 32 (most notably: a space or CR/LF) is detected.

Server Hostname

The DNS or IP address given on the command line

Server Port

The port to which ab is connecting. If no port is given on the command line, this will default to 80 for http and 443 for https.

SSL/TLS Protocol

The protocol parameters negotiated between the client and server. This will only be printed if SSL is used.

Document Path

The request URI parsed from the command line string.

Document Length

This is size in bytes of the first successfully returning document. If the document length changes during testing, the response is considered an error.

Concurrency Level

The number of concurrent clients used during the test

Time taken for tests

This is the time taken from the moment the first socket connection is created to the moment the last response is received

Complete requests

The number of successful responses received

Failed requests

The number of requests that were considered a failure. If the number

is greater than zero, another line will be printed showing the number of requests that failed due to connecting, reading, incorrect content length, or exceptions.

Write errors

The number of errors that failed during write (broken pipe).

Non-2xx responses

The number of responses that were not in the 200 series of response codes. If all responses were 200, this field is not printed.

Keep-Alive requests

The number of connections that resulted in Keep-Alive requests

Total body sent

If configured to send data as part of the test, this is the total number of bytes sent during the tests. This field is omitted if the test did not include a body to send.

Total transferred

The total number of bytes received from the server. This number is essentially the number of bytes sent over the wire.

HTML transferred

The total number of document bytes received from the server. This number excludes bytes received in HTTP headers

Requests per second

This is the number of requests per second. This value is the result of dividing the number of requests by the total time taken

Time per request

The average time spent per request. The first value is calculated with the formula $\text{concurrency} * \text{timetaken} * 1000 / \text{done}$ while the second value is calculated with the formula $\text{timetaken} * 1000 / \text{done}$

Transfer rate

The rate of transfer as calculated by the formula $\text{totalread} /$

1024 / timetaken

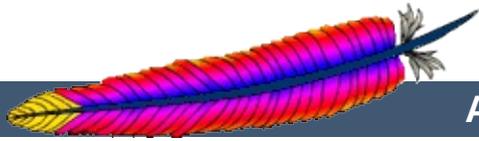


There are various statically declared buffers of fixed length. Combined with the lazy parsing of the command line arguments, the response headers from the server and other external inputs, this might bite you.

It does not implement HTTP/1.x fully; only accepts some 'expected' forms of responses. The rather heavy use of `strstr(3)` shows up top in profile, which might indicate a performance problem; *i.e.*, you would measure the `ab` performance rather than the server's.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

apachectl - Apache HTTP Server Control Interface

`apachectl` is a front end to the Apache HyperText Transfer Protocol (HTTP) server. It is designed to help the administrator control the functioning of the Apache [httpd](#) daemon.

The `apachectl` script can operate in two modes. First, it can act as a simple front-end to the [httpd](#) command that simply sets any necessary environment variables and then invokes [httpd](#), passing through any command line arguments. Second, `apachectl` can act as a SysV init script, taking simple one-word arguments like `start`, `restart`, and `stop`, and translating them into appropriate signals to [httpd](#).

If your Apache installation uses non-standard paths, you will need to edit the `apachectl` script to set the appropriate paths to the [httpd](#) binary. You can also specify any necessary [httpd](#) command line arguments. See the comments in the script for details.

The `apachectl` script returns a 0 exit value on success, and >0 if an error occurs. For more details, view the comments in the script.

See also

- [Starting Apache](#)
- [Stopping Apache](#)
- [Configuration Files](#)
- [Platform Docs](#)
- [httpd](#)



Synopsis

When acting in pass-through mode, `apachectl` can take all the arguments available for the [httpd](#) binary.

apachectl [*httpd-argument*]

When acting in SysV init mode, `apachectl` takes simple, one-word commands, defined below.

apachectl *command*



Only the SysV init-style options are defined here. Other arguments are defined on the [httpd](#) manual page.

start

Start the Apache [httpd](#) daemon. Gives an error if it is already running. This is equivalent to `apachectl -k start`.

stop

Stops the Apache [httpd](#) daemon. This is equivalent to `apachectl -k stop`.

restart

Restarts the Apache [httpd](#) daemon. If the daemon is not running, it is started. This command automatically checks the configuration files as in `configtest` before initiating the restart to make sure the daemon doesn't die. This is equivalent to `apachectl -k restart`.

fullstatus

Displays a full status report from [mod_status](#). For this to work, you need to have [mod_status](#) enabled on your server and a text-based browser such as `lynx` available on your system. The URL used to access the status report can be set by editing the `STATUSURL` variable in the script.

status

Displays a brief status report. Similar to the `fullstatus` option, except that the list of requests currently being served is omitted.

graceful

Gracefully restarts the Apache [httpd](#) daemon. If the daemon is not running, it is started. This differs from a normal restart in that currently open connections are not aborted. A side effect is that old log files will not be closed immediately. This means that if used in a log rotation script, a substantial delay may be necessary to ensure

that the old log files are closed before processing them. This command automatically checks the configuration files as in `configtest` before initiating the restart to make sure Apache doesn't die. This is equivalent to `apachectl -k graceful`.

graceful-stop

Gracefully stops the Apache `httpd` daemon. This differs from a normal stop in that currently open connections are not aborted. A side effect is that old log files will not be closed immediately. This is equivalent to `apachectl -k graceful-stop`.

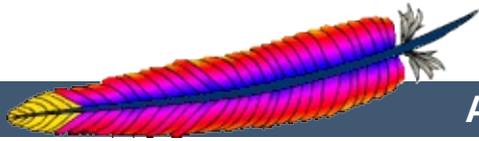
configtest

Run a configuration file syntax test. It parses the configuration files and either reports `Syntax Ok` or detailed information about the particular syntax error. This is equivalent to `apachectl -t`.

The following option was available in earlier versions but has been removed.

startssl

To start `httpd` with SSL support, you should edit your configuration file to include the relevant directives and then use the normal `apachectl start`.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

apxs - APache eXtenSion tool

apxs is a tool for building and installing extension modules for the Apache HyperText Transfer Protocol (HTTP) server. This is achieved by building a dynamic shared object (DSO) from one or more source or object *files* which then can be loaded into the Apache server under runtime via the `LoadModule` directive from `mod_so`.

So to use this extension mechanism your platform has to support the DSO feature and your Apache `httpd` binary has to be built with the `mod_so` module. The apxs tool automatically complains if this is not the case. You can check this yourself by manually running the command

```
$ httpd -l
```

The module `mod_so` should be part of the displayed list. If these requirements are fulfilled you can easily extend your Apache server's functionality by installing your own modules with the DSO mechanism by the help of this apxs tool:

```
$ apxs -i -a -c mod_foo.c
gcc -fpic -DSHARED_MODULE -I/path/to/apache/include -c mod_foo.c
ld -Bshareable -o mod_foo.so mod_foo.o
cp mod_foo.so /path/to/apache/modules/mod_foo.so
chmod 755 /path/to/apache/modules/mod_foo.so
[activating module `foo' in /path/to/apache/etc/httpd.conf]
$ apachectl restart
/path/to/apache/sbin/apachectl restart: httpd not running, trying
to start
[Tue Mar 31 11:27:55 1998] [debug] mod_so.c(303): loaded module
foo_module
/path/to/apache/sbin/apachectl restart: httpd started
$ _
```

The arguments *files* can be any C source file (.c), a object file (.o) or even a library archive (.a). The apxs tool automatically recognizes

these extensions and automatically used the C source files for compilation while just using the object and archive files for the linking phase. But when using such pre-compiled objects make sure they are compiled for position independent code (PIC) to be able to use them for a dynamically loaded shared object. For instance with GCC you always just have to use `-fpic`. For other C compilers consult its manual page or at watch for the flags `apxs` uses to compile the object files.

For more details about DSO support in Apache read the documentation of [mod_so](#) or perhaps even read the `src/modules/standard/mod_so.c` source file.

See also

[apachectl](#)

[httpd](#)



Synopsis

apxs -g [**-S** *name=value*] **-n** *modname*

apxs -q [**-S** *name=value*] *query* ...

apxs -c [**-S** *name=value*] [**-o** *dsofile*] [**-I** *incdir*]
[**-D** *name=value*] [**-L** *libdir*] [**-l** *libname*] [**-Wc**,*compiler-flags*] [**-Wl**,*linker-flags*] *files* ...

apxs -i [**-S** *name=value*] [**-n** *modname*] [**-a**] [**-A**]
dso-file ...

apxs -e [**-S** *name=value*] [**-n** *modname*] [**-a**] [**-A**]
dso-file ...



Common Options

-n *modname*

This explicitly sets the module name for the **-i** (install) and **-g** (template generation) option. Use this to explicitly specify the module name. For option **-g** this is required, for option **-i** the `apxs` tool tries to determine the name from the source or (as a fallback) at least by guessing it from the filename.

Query Options

-q

Performs a query for `apxs`'s knowledge about certain settings. The *query* parameters can be one or more of the following strings: `CC`, `CFLAGS`, `CFLAGS_SHLIB`, `INCLUDEDIR`, `LD_SHLIB`, `LDFLAGS_SHLIB`, `LIBEXECDIR`, `LIBS_SHLIB`, `SBINDIR`, `SYSCONFDIR`, `TARGET`.

Use this for manually determining settings. For instance use

```
INC=-I`apxs -q INCLUDEDIR`
```

inside your own Makefiles if you need manual access to Apache's C header files.

Configuration Options

-S *name=value*

This option changes the `apxs` settings described above.

Template Generation Options

-g

This generates a subdirectory *name* (see option **-n**) and there two files: A sample module source file named `mod_name.c` which can

be used as a template for creating your own modules or as a quick start for playing with the apxs mechanism. And a corresponding Makefile for even easier build and installing of this module.

DSO Compilation Options

-c

This indicates the compilation operation. It first compiles the C source files (.c) of *files* into corresponding object files (.o) and then builds a dynamically shared object in *dsofile* by linking these object files plus the remaining object files (.o and .a) of *files*. If no -o option is specified the output file is guessed from the first filename in *files* and thus usually defaults to `mod_name.so`.

-o *dsofile*

Explicitly specifies the filename of the created dynamically shared object. If not specified and the name cannot be guessed from the *files* list, the fallback name `mod_unknown.so` is used.

-D *name=value*

This option is directly passed through to the compilation command(s). Use this to add your own defines to the build process.

-I *incdir*

This option is directly passed through to the compilation command(s). Use this to add your own include directories to search to the build process.

-L *libdir*

This option is directly passed through to the linker command. Use this to add your own library directories to search to the build process.

-l *libname*

This option is directly passed through to the linker command. Use this to add your own libraries to search to the build process.

-Wc, *compiler-flags*

This option passes *compiler-flags* as additional flags to the `libtool`

`--mode=compile` command. Use this to add local compiler-specific options.

`-wl, linker-flags`

This option passes *linker-flags* as additional flags to the `libtool -mode=link` command. Use this to add local linker-specific options.

DSO Installation and Configuration Options

`-i`

This indicates the installation operation and installs one or more dynamically shared objects into the server's *modules* directory.

`-a`

This activates the module by automatically adding a corresponding [LoadModule](#) line to Apache's `httpd.conf` configuration file, or by enabling it if it already exists.

`-A`

Same as option `-a` but the created [LoadModule](#) directive is prefixed with a hash sign (`#`), *i.e.*, the module is just prepared for later activation but initially disabled.

`-e`

This indicates the editing operation, which can be used with the `-a` and `-A` options similarly to the `-i` operation to edit Apache's `httpd.conf` configuration file without attempting to install the module.



Assume you have an Apache module named `mod_foo.c` available which should extend Apache's server functionality. To accomplish this you first have to compile the C source into a shared object suitable for loading into the Apache server under runtime via the following command:

```
$ apxs -c mod_foo.c
/path/to/libtool --mode=compile gcc ... -c mod_foo.c
/path/to/libtool --mode=link gcc ... -o mod_foo.la mod_foo.slo
$ _
```

Then you have to update the Apache configuration by making sure a [LoadModule](#) directive is present to load this shared object. To simplify this step `apxs` provides an automatic way to install the shared object in its "modules" directory and updating the `httpd.conf` file accordingly. This can be achieved by running:

```
$ apxs -i -a mod_foo.la
/path/to/instldso.sh mod_foo.la /path/to/apache/modules
/path/to/libtool --mode=install cp mod_foo.la /path/to/apache/modules
... chmod 755 /path/to/apache/modules/mod_foo.so
[activating module `foo' in /path/to/apache/conf/httpd.conf]
$ _
```

This way a line named

```
LoadModule foo_module modules/mod_foo.so
```

is added to the configuration file if still not present. If you want to have this disabled per default use the `-A` option, *i.e.*

```
$ apxs -i -A mod_foo.c
```

For a quick test of the `apxs` mechanism you can create a sample Apache module template plus a corresponding Makefile via:

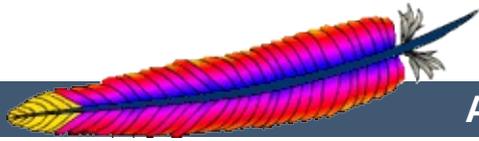
```
$ apxs -g -n foo
Creating [DIR] foo
Creating [FILE] foo/Makefile
Creating [FILE] foo/modules.mk
Creating [FILE] foo/mod_foo.c
Creating [FILE] foo/.deps
$ _
```

Then you can immediately compile this sample module into a shared object and load it into the Apache server:

```
$ cd foo
$ make all reload
apxs -c mod_foo.c
/path/to/libtool --mode=compile gcc ... -c mod_foo.c
/path/to/libtool --mode=link gcc ... -o mod_foo.la mod_foo.slo
apxs -i -a -n "foo" mod_foo.la
/path/to/instldso.sh mod_foo.la /path/to/apache/modules
/path/to/libtool --mode=install cp mod_foo.la /path/to/apache/modules
... chmod 755 /path/to/apache/modules/mod_foo.so
[activating module `foo' in /path/to/apache/conf/httpd.conf]
apachectl restart
/path/to/apache/sbin/apachectl restart: httpd not running, trying to
start
[Tue Mar 31 11:27:55 1998] [debug] mod_so.c(303): loaded module
foo_module
/path/to/apache/sbin/apachectl restart: httpd started
$ _
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

configure - Configure the source tree

The `configure` script configures the source tree for compiling and installing the Apache HTTP Server on your particular platform. Various options allow the compilation of a server corresponding to your personal requirements.

This script, included in the root directory of the source distribution, is for compilation on Unix and Unix-like systems only. For other platforms, see the [platform](#) documentation.

See also

[Compiling and Installing](#)



Synopsis

You should call the `configure` script from within the root directory of the distribution.

`./configure` [*OPTION*]... [*VAR=VALUE*]...

To assign environment variables (e.g. `CC`, `CFLAGS` ...), specify them as `VAR=VALUE`. See [below](#) for descriptions of some of the useful variables.



- [Configuration options](#)
- [Installation directories](#)
- [System types](#)
- [Optional features](#)
- [Options for support programs](#)

Configuration options

The following options influence the behavior of `configure` itself.

-C

--config-cache

This is an alias for `--cache-file=config.cache`

--cache-file=FILE

The test results will be cached in file *FILE*. This option is disabled by default.

-h

--help [short|recursive]

Output the help and exit. With the argument `short` only options specific to this package will be displayed. The argument `recursive` displays the short help of all the included packages.

-n

--no-create

The `configure` script is run normally but does not create output files. This is useful to check the test results before generating makefiles for compilation.

-q

--quiet

Do not print checking ... messages during the `configure` process.

--srcdir=*DIR*

Defines directory *DIR* to be the source file directory. Default is the directory where `configure` is located, or the parent directory.

--silent

Same as `--quiet`

-V

--version

Display copyright information and exit.

Installation directories

These options define the installation directory. The installation tree depends on the selected layout.

--prefix=*PREFIX*

Install architecture-independent files in *PREFIX*. By default the installation directory is set to `/usr/local/apache2`.

--exec-prefix=*EPREFIX*

Install architecture-dependent files in *EPREFIX*. By default the installation directory is set to the *PREFIX* directory.

By default, `make install` will install all the files in `/usr/local/apache2/bin`, `/usr/local/apache2/lib` etc. You can specify an installation prefix other than `/usr/local/apache2` using `--prefix`, for instance `--prefix=$HOME`.

Define a directory layout

--enable-layout=*LAYOUT*

Configure the source code and build scripts to assume an installation tree based on the layout *LAYOUT*. This allows you to separately specify the locations for each type of file within the Apache HTTP Server installation. The `config.layout` file contains several example configurations, and you can also create your own custom

configuration following the examples. The different layouts in this file are grouped into `<Layout F00> . . . </Layout>` sections and referred to by name as in F00. The default layout is Apache.

Fine tuning of the installation directories

For better control of the installation directories, use the options below. Please note that the directory defaults are set by `autoconf` and are overwritten by the corresponding layout setting.

--bindir=DIR

Install user executables in *DIR*. The user executables are supporting programs like [htpasswd](#), [dbmmanage](#), etc. which are useful for site administrators. By default *DIR* is set to *EPREFIX/bin*.

--datadir=DIR

Install read-only architecture-independent data in *DIR*. By default *datadir* is set to *PREFIX/share*. This option is offered by `autoconf` and currently unused.

--includedir=DIR

Install C header files in *DIR*. By default *includedir* is set to *EPREFIX/include*.

--infodir=DIR

Install info documentation in *DIR*. By default *infodir* is set to *PREFIX/info*. This option is currently unused.

--libdir=DIR

Install object code libraries in *DIR*. By default *libdir* is set to *EPREFIX/lib*.

--libexecdir=DIR

Install the program executables (i.e., shared modules) in *DIR*. By default *libexecdir* is set to *EPREFIX/modules*.

--localstatedir=DIR

Install modifiable single-machine data in *DIR*. By default

`localstatedir` is set to *PREFIX/var*. This option is offered by `autoconf` and currently unused.

--mandir=DIR

Install the man documentation in *DIR*. By default `mandir` is set to *EPREFIX/man*.

--oldincludedir=DIR

Install C header files for non-gcc in *DIR*. By default `oldincludedir` is set to */usr/include*. This option is offered by `autoconf` and currently unused.

--sbindir=DIR

Install the system administrator executables in *DIR*. Those are server programs like [httpd](#), [apachectl](#), [suexec](#), etc. which are necessary to run the Apache HTTP Server. By default `sbindir` is set to *EPREFIX/sbin*.

--sharedstatedir=DIR

Install modifiable architecture-independent data in *DIR*. By default `sharedstatedir` is set to *PREFIX/com*. This option is offered by `autoconf` and currently unused.

--sysconfdir=DIR

Install read-only single-machine data like the server configuration files `httpd.conf`, `mime.types`, etc. in *DIR*. By default `sysconfdir` is set to *PREFIX/conf*.

System types

These options are used to cross-compile the Apache HTTP Server to run on another system. In normal cases, when building and running the server on the same system, these options are not used.

--build=BUILD

Defines the system type of the system on which the tools are being built. It defaults to the result of the script `config.guess`.

--host=HOST

Defines the system type of the system on which the server will run. *HOST* defaults to *BUILD*.

--target=TARGET

Configure for building compilers for the system type *TARGET*. It defaults to *HOST*. This option is offered by autoconf and not necessary for the Apache HTTP Server.

Optional Features

These options are used to fine tune the features your HTTP server will have.

General syntax

Generally you can use the following syntax to enable or disable a feature:

--disable-*FEATURE*

Do not include *FEATURE*. This is the same as `--enable-FEATURE=no`.

--enable-*FEATURE*[=*ARG*]

Include *FEATURE*. The default value for *ARG* is yes.

--enable-*MODULE*=shared

The corresponding module will be build as DSO module.

--enable-*MODULE*=static

By default enabled modules are linked statically. You can force this explicitly.

Note

`configure` will not complain about `--enable-foo` even if *foo* doesn't exist, so you need to type carefully.

Modules enabled by default

Some modules are compiled by default and have to be disabled explicitly. Use the following options to remove discrete modules from the compilation process.

--disable-actions

Disable action triggering on requests, which is provided by [mod_actions](#).

--disable-alias

Disable the mapping of requests to different parts of the filesystem, which is provided by [mod_alias](#).

--disable-asis

Disable support for as-is filetypes, which is provided by [mod_asis](#).

--disable-auth-basic

Disable user-based access control provided by [mod_auth_basic](#). This module provides for HTTP Basic Authentication, where the usernames and passwords are stored in plain text files.

--disable-autoindex

Disable the directory listing functionality provided by [mod_autoindex](#).

--disable-cgi

[mod_cgi](#), which provides support for CGI scripts, is enabled by default when using a non-threaded MPM. Use this option to disable CGI support.

--disable-cgid

When using the threaded MPMs [worker](#) support for CGI scripts is provided by [mod_cgid](#) by default. To disable CGI support use this option.

--disable-charset-lite

Disable character set translation provided by [mod_charset_lite](#). This module will be installed by default only on EBCDIC systems.

--disable-dir

Disable directory request handling provided by [mod_dir](#).

--disable-env

Enable setting and clearing of environment variables, which is provided by [mod_env](#).

--disable-http

Disable the HTTP protocol handling. The `http` module is a basic one, enabling the server to function as an HTTP server. It is only useful to disable it if you want to use another protocol module instead. **Don't disable this module unless you are really sure what you are doing.**

Note: This module will always be linked statically.

--disable-imagemap

Disable support for server based imagemaps, which provided by [mod_imagemap](#).

--disable-include

Disable Server Side Includes provided by [mod_include](#).

--disable-log-config

Disable the logging configuration provided by [mod_log_config](#). You won't be able to log requests to the server without this module.

--disable-mime

[mod_mime](#) associates the requested filename's extensions with the file's behavior and content ([mime-type](#), language, character set and encoding). Disabling this module is normally not recommended.

--disable-negotiation

Disable content negotiation provided by [mod_negotiation](#).

--disable-setenvif

Disable support for basing environment variables on headers, which is provided by [mod_setenvif](#).

--disable-status

Enable the process/thread monitoring, which is provided by

[mod_status](#).

--disable-userdir

Disable the mapping of requests to user-specific directories, which is provided by [mod_userdir](#).

Modules, disabled by default

Some modules are compiled by default and have to be enabled explicitly or by using the keywords `most` or `all` (see `--enable-mods-shared` below for further explanation) to be available. Therefore use the options below.

--enable-authn-anon

Enable anonymous user access provided by [mod_authn_anon](#).

--enable-authn-dbm

[mod_authn_dbm](#) provides for HTTP Basic Authentication, where the usernames and passwords are stored in DBM type database files. Use this option to enable the module.

--enable-authz-dbm

[mod_authz_dbm](#) provides for HTTP Basic Authorization, where the usernames and groups are stored in DBM type database files. Use this option to enable the module.

--enable-auth-digest

Enable RFC2617 Digest authentication provided by [mod_auth_digest](#). This module uses plain text files to store the credentials.

--enable-authnz-ldap

Enable LDAP based authentication provided by [mod_authnz_ldap](#).

--enable-cache

Enable dynamic file caching provided by [mod_cache](#). This experimental module may be interesting for servers with high load or caching proxy servers. At least one storage management module

(e.g. [mod_disk_cache](#) or [mod_mem_cache](#)) is also necessary.

--enable-cern-meta

Enable the CERN-type meta files support provided by [mod_cern_meta](#).

--enable-charset-lite

Enable character set translation provided by [mod_charset_lite](#). This module will be installed by default only on EBCDIC systems. On other systems, you have to enable it.

--enable-dav

Enable the WebDAV protocol handling provided by [mod_dav](#). Support for filesystem resources is provided by the separate module [mod_dav_fs](#). This module is also automatically enabled with --enable-dav.

Note: [mod_dav](#) can only be used together with the http protocol module.

--enable-dav-fs

Enable DAV support for filesystem resources, which is provided by [mod_dav_fs](#). This module is a provider for the [mod_dav](#) module, so you should also use --enable-dav.

--enable-dav-lock

Enable [mod_dav_lock](#) which provides generic DAV locking support for backend modules. This module needs at least [mod_dav](#) to function, so you should also use --enable-dav.

--enable-deflate

Enable deflate transfer encoding provided by [mod_deflate](#).

--enable-disk-cache

Enable disk caching provided by [mod_disk_cache](#).

--enable-expires

Enable Expires header control provided by [mod_expires](#).

--enable-ext-filter

Enable the external filter support provided by [mod_ext_filter](#).

--enable-file-cache

Enable the file cache provided by [mod_file_cache](#).

--enable-headers

Enable control of HTTP headers provided by [mod_headers](#).

--enable-info

Enable the server information provided by [mod_info](#).

--enable-ldap

Enable LDAP caching and connection pooling services provided by [mod_ldap](#).

--enable-logio

Enable logging of input and output bytes including headers provided by [mod_logio](#).

--enable-mem-cache

Enable memory caching provided by [mod_mem_cache](#).

--enable-mime-magic

Enable automatical determining of [MIME types](#), which is provided by [mod_mime_magic](#).

--enable-isapi

Enable the isapi extension support provided by [mod_isapi](#).

--enable-proxy

Enable the proxy/gateway functionality provided by [mod_proxy](#). The proxying capabilities for AJP13, CONNECT, FTP, HTTP and the balancer are provided by the separate modules [mod_proxy_ajp](#), [mod_proxy_connect](#), [mod_proxy_ftp](#), [mod_proxy_http](#) and [mod_proxy_balancer](#). These five modules are also automatically enabled with --enable-proxy.

--enable-proxy-ajp

Enable proxy support for AJP13 (Apache JServ Protocol

1.3) request handling, which is provided by [mod_proxy_ajp](#). This module is an extension for the [mod_proxy](#) module, so you should also use `--enable-proxy`.

--enable-proxy-balancer

Enable load balancing support for the AJP13, FTP and HTTP protocols, which is provided by [mod_proxy_balancer](#). This module is an extension for the [mod_proxy](#) module, so you should also use `--enable-proxy`.

--enable-proxy-connect

Enable proxy support for CONNECT request handling, which is provided by [mod_proxy_connect](#). This module is an extension for the [mod_proxy](#) module, so you should also use `--enable-proxy`.

--enable-proxy-ftp

Enable proxy support for FTP requests, which is provided by [mod_proxy_ftp](#). This module is an extension for the [mod_proxy](#) module, so you should also use `--enable-proxy`.

--enable-proxy-http

Enable proxy support for HTTP requests, which is provided by [mod_proxy_http](#). This module is an extension for the [mod_proxy](#) module, so you should also use `--enable-proxy`.

--enable-rewrite

Enable rule based URL manipulation provided by [mod_rewrite](#).

--enable-so

Enable DSO capability provided by [mod_so](#). This module will be automatically enabled if you use the `--enable-mods-shared` option.

--enable-spelling

Enable the functionality to correct common URL misspellings, which is provided by [mod_spelling](#).

--enable-ssl

Enable support for SSL/TLS provided by [mod_ssl](#).

--enable-unique-id

Enable the generation of per-request unique ids, which is provided by [mod_unique_id](#).

--enable-usertrack

Enable user-session tracking provided by [mod_usertrack](#).

--enable-vhost-alias

Enable mass virtual hosting provided by [mod_vhost_alias](#).

Modules for developers

The following modules are useful only for developers and testing purposes and are disabled by default. Use the following options to enable them. If you are not sure whether you need one of these modules, omit them.

--enable-bucketeer

Enable the manipulation filter for buckets, which is provided by `mod_bucketeer`.

--enable-case-filter

Enable the example uppercase conversion output filter support of `mod_case_filter`.

--enable-case-filter-in

Enable the example uppercase conversion input filter support of `mod_case_filter_in`.

--enable-echo

Enable the ECHO server provided by [mod_echo](#).

--enable-example

Enable the example and demo module [mod_example](#).

--enable-optional-fn-export

Enable the example for an optional function exporter, which is

provided by `mod_optional_fn_export`.

--enable-optional-fn-import

Enable the example for an optional function importer, which is provided by `mod_optional_fn_import`.

--enable-optional-hook-export

Enable the example for an optional hook exporter, which is provided by `mod_optional_hook_export`.

--enable-optional-hook-import

Enable the example optional hook importer, which is provided by `mod_optional_hook_import`.

MPMs and third-party modules

To add the necessary Multi Processing Module and additional third-party modules use the following options:

--with-module=*module-type:module-file*[, *module-type:module-file*]

Add one or more third-party modules to the list of statically linked modules. The module source file *module-file* will be searched in the `modules/module-type` subdirectory of your Apache HTTP server source tree. If it is not found there `configure` is considering *module-file* to be an absolute file path and tries to copy the source file into the *module-type* subdirectory. If the subdirectory doesn't exist it will be created and populated with a standard `Makefile.in`.

This option is useful to add small external modules consisting of one source file. For more complex modules you should read the vendor's documentation.

Note

If you want to build a DSO module instead of a statically linked use [apxs](#).

--with-mpm=MPM

Choose the process model for your server. You have to select exactly one [Multi-Processing Module](#). Otherwise the [default MPM](#) for your operating system will be taken. Possible MPMs are [beos](#), [mpmt_os2](#), [prefork](#), and [worker](#).

Cumulative and other options

--enable-maintainer-mode

Turn on debugging and compile time warnings.

--enable-mods-shared=MODULE-LIST

Defines a list of modules to be enabled and build as dynamic shared modules. This mean, these module have to be loaded dynamically by using the [LoadModule](#) directive.

MODULE-LIST is a space separated list of modulenames enclosed by quotation marks. The module names are given without the preceding `mod_`. For example:

```
--enable-mods-shared='headers rewrite dav'
```

Additionally you can use the special keywords `all` and `most`. For example,

```
--enable-mods-shared=most
```

will compile most modules and build them as DSO modules.

Caveat: `--enable-mods-shared=all` does not actually build all modules. To build all modules then, one might use:

```
./configure \  
  --with-ldap \  
  --enable-mods-shared="all ssl ldap cache proxy authn_alias  
  mem_cache file_cache authnz_ldap charset_lite dav_lock  
  disk_cache"
```

--enable-modules=MODULE-LIST

This option behaves similar to `--enable-mods-shared`, but will link the given modules statically. This means, these modules will always be present while running `httpd`. They need not be loaded with `LoadModule`.

--enable-v4-mapped

Allow IPv6 sockets to handle IPv4 connections.

--with-port=PORT

This defines the port on which `httpd` will listen. This port number is used when generating the configuration file `httpd.conf`. The default is 80.

--with-program-name

Define an alternative executable name. The default is `httpd`.

Optional packages

These options are used to define optional packages.

General syntax

Generally you can use the following syntax to define an optional package:

--with-PACKAGE[=ARG]

Use the package `PACKAGE`. The default value for `ARG` is `yes`.

--without-PACKAGE

Do not use the package `PACKAGE`. This is the same as `--with-PACKAGE=no`. This option is provided by `autoconf` but not very useful for the Apache HTTP Server.

Specific packages

--with-apr=DIR|FILE

The [Apache Portable Runtime](#) (APR) is part of the httpd source distribution and will automatically be build together with the HTTP server. If you want to use an already installed APR instead you have to tell `configure` the path to the `apr-config` script. You may set the absolute path and name or the directory to the installed APR. `apr-config` must exist within this directory or the subdirectory `bin`.

--with-apr-util=DIR|FILE

The Apache Portable Runtime Utilities (APU) are part of the httpd source distribution and will automatically be build together with the HTTP server. If you want to use an already installed APU instead you have to tell `configure` the path to the `apu-config` script. You may set the absolute path and name or the directory to the installed APU. `apu-config` must exist within this directory or the subdirectory `bin`.

--with-ssl=DIR

If [mod_ssl](#) has been enabled `configure` searches for an installed OpenSSL. You can set the directory path to the SSL/TLS toolkit instead.

--with-z=DIR

`configure` searches automatically for an installed `zlib` library if your source configuration requires one (e.g., when [mod_deflate](#) is enabled). You can set the directory path to the compression library instead.

Several features of the Apache HTTP Server, including [mod_authn_dbm](#), [mod_authz_dbm](#), and [mod_rewrite](#)'s [DBM RewriteMap](#) use simple key/value databases for quick lookups of information. SDBM is included in the APU, so this database is always available. If you would like to use other database types, use the following options to enable them:

--with-gdbm[=path]

If no `path` is specified, `configure` will search for the include files

and libraries of a GNU DBM installation in the usual search paths. An explicit *path* will cause `configure` to look in *path/lib* and *path/include* for the relevant files. Finally, the *path* may specify specific include and library paths separated by a colon.

--with-ndbm[=*path*]

Like `--with-gdbm`, but searches for a New DBM installation.

--with-berkeley-db[=*path*]

Like `--with-gdbm`, but searches for a Berkeley DB installation.

Note

The DBM options are provided by the APU and passed through to its configuration script. They are useless when using an already installed APU defined by `--with-apr-util`.

You may use more than one DBM implementation together with your HTTP server. The appropriated DBM type will be configured within the runtime configuration at each time.

Options for support programs

--enable-static-support

Build a statically linked version of the support binaries. This means, a stand-alone executable will be built with all the necessary libraries integrated. Otherwise the support binaries are linked dynamically by default.

--enable-suexec

Use this option to enable [suexec](#), which allows you to set uid and gid for spawned processes. **Do not use this option unless you understand all the security implications of running a suid binary on your server.** Further options to configure [suexec](#) are described [below](#).

It is possible to create a statically linked binary of a single support

program by using the following options:

--enable-static-ab

Build a statically linked version of [ab](#).

--enable-static-checkgid

Build a statically linked version of [checkgid](#).

--enable-static-htdbm

Build a statically linked version of [htdbm](#).

--enable-static-htdigest

Build a statically linked version of [htdigest](#).

--enable-static-htpasswd

Build a statically linked version of [htpasswd](#).

--enable-static-logresolve

Build a statically linked version of [logresolve](#).

--enable-static-rotatelogs

Build a statically linked version of [rotatelogs](#).

suexec configuration options

The following options are used to fine tune the behavior of [suexec](#). See [Configuring and installing suEXEC](#) for further information.

--with-suexec-bin

This defines the path to [suexec](#) binary. Default is `--sbindir` (see [Fine tuning of installation directories](#)).

--with-suexec-caller

This defines the user allowed to call [suexec](#). It should be the same as the user under which [httpd](#) normally runs.

--with-suexec-docroot

This defines the directory tree under which [suexec](#) access is allowed for executables. Default value is `--datadir/htdocs`.

--with-suexec-gidmin

Define this as the lowest GID allowed to be a target user for [suexec](#). The default value is 100.

--with-suexec-logfile

This defines the filename of the [suexec](#) logfile. By default the logfile is named suexec_log and located in --logfiledir.

--with-suexec-safepath

Define the value of the environment variable PATH to be set for processes started by [suexec](#). Default value is /usr/local/bin:/usr/bin:/bin.

--with-suexec-userdir

This defines the subdirectory under the user's directory that contains all executables for which [suexec](#) access is allowed. This setting is necessary when you want to use [suexec](#) together with user-specific directories (as provided by [mod_userdir](#)). The default is public_html.

--with-suexec-uidmin

Define this as the lowest UID allowed to be a target user for [suexec](#). The default value is 100.

--with-suexec-umask

Set umask for processes started by [suexec](#). It defaults to your system settings.



ENVIRONMENT VARIABLES

There are some useful environment variables to override the choices made by `configure` or to help it to find libraries and programs with nonstandard names or locations.

CC

Define the C compiler command to be used for compilation.

CFLAGS

Set C compiler flags you want to use for compilation.

CPP

Define the C preprocessor command to be used.

CPPFLAGS

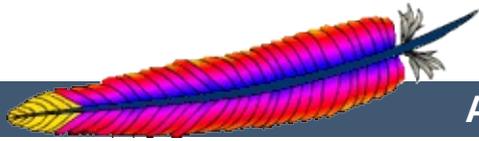
Set C/C++ preprocessor flags, e.g. `-Iincludedir` if you have headers in a nonstandard directory `includedir`.

LDFLAGS

Set linker flags, e.g. `-Llibdir` if you have libraries in a nonstandard directory `libdir`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

dbmmanage - Manage user authentication files in DBM format

dbmmanage is used to create and update the DBM format files used to store usernames and password for basic authentication of HTTP users via [mod_authn_dbm](#). Resources available from the Apache HTTP server can be restricted to just the users listed in the files created by dbmmanage. This program can only be used when the usernames are stored in a DBM file. To use a flat-file database see [htpasswd](#).

This manual page only lists the command line arguments. For details of the directives necessary to configure user authentication in [httpd](#) see the httpd manual, which is part of the Apache distribution or can be found at <http://httpd.apache.org/>.

See also

[httpd](#)

[mod_authn_dbm](#)

[mod_authz_dbm](#)



Synopsis

```
dbmmanage [ encoding ] filename  
add|adduser|check|delete|update username [ encpasswd [ group[,group...] [ comment ] ] ]
```

```
dbmmanage filename view [ username ]
```

```
dbmmanage filename import
```



filename

The filename of the DBM format file. Usually without the extension .db, .pag, or .dir.

username

The user for which the operations are performed. The *username* may not contain a colon (:).

encpasswd

This is the already encrypted password to use for the update and add commands. You may use a hyphen (-) if you want to get prompted for the password, but fill in the fields afterwards. Additionally when using the update command, a period (.) keeps the original password untouched.

group

A group, which the user is member of. A groupname may not contain a colon (:). You may use a hyphen (-) if you don't want to assign the user to a group, but fill in the comment field. Additionally when using the update command, a period (.) keeps the original groups untouched.

comment

This is the place for your opaque comments about the user, like realname, mailaddress or such things. The server will ignore this field.

Encodings

-d

crypt encryption (default, except on Win32, Netware)

-m

MD5 encryption (default on Win32, Netware)

-s

SHA1 encryption

-p

plaintext (*not recommended*)

Commands

add

Adds an entry for *username* to *filename* using the encrypted password *encpasswd*.

```
dbmmanage passwords.dat add rbowen foKntnEF3KSXA
```

adduser

Asks for a password and then adds an entry for *username* to *filename*.

```
dbmmanage passwords.dat adduser krietz
```

check

Asks for a password and then checks if *username* is in *filename* and if its password matches the specified one.

```
dbmmanage passwords.dat check rbowen
```

delete

Deletes the *username* entry from *filename*.

```
dbmmanage passwords.dat delete rbowen
```

import

Reads *username:password* entries (one per line) from STDIN and adds them to *filename*. The passwords already have to be crypted.

update

Same as the adduser command, except that it makes sure

username already exists in *filename*.

```
dbmmanage passwords.dat update rbowen
```

view

Just displays the contents of the DBM file. If you specify a *username*, it displays the particular record only.

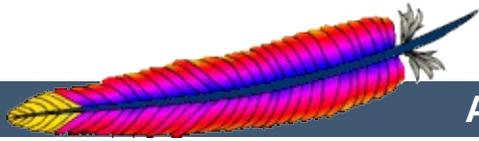
```
dbmmanage passwords.dat view
```



One should be aware that there are a number of different DBM file formats in existence, and with all likelihood, libraries for more than one format may exist on your system. The three primary examples are SDBM, NDBM, the GNU project's GDBM, and Berkeley DB 2. Unfortunately, all these libraries use different file formats, and you must make sure that the file format used by *filename* is the same format that `dbmmanage` expects to see. `dbmmanage` currently has no way of determining what type of DBM file it is looking at. If used against the wrong format, will simply return nothing, or may create a different DBM file with a different name, or at worst, it may corrupt the DBM file if you were attempting to write to it.

`dbmmanage` has a list of DBM format preferences, defined by the `@AnyDBM: :ISA` array near the beginning of the program. Since we prefer the Berkeley DB 2 file format, the order in which `dbmmanage` will look for system libraries is Berkeley DB 2, then NDBM, then GDBM and then SDBM. The first library found will be the library `dbmmanage` will attempt to use for all DBM file transactions. This ordering is slightly different than the standard `@AnyDBM: :ISA` ordering in Perl, as well as the ordering used by the simple `dbmopen()` call in Perl, so if you use any other utilities to manage your DBM files, they must also follow this preference ordering. Similar care must be taken if using programs in other languages, like C, to access these files.

One can usually use the `file` program supplied with most Unix systems to see what format a DBM file is in.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

htcacheclean - Clean up the disk cache

htcacheclean is used to keep the size of [mod_disk_cache](#)'s storage within a certain limit. This tool can run either manually or in daemon mode. When running in daemon mode, it sleeps in the background and checks the cache directories at regular intervals for cached content to be removed. You can stop the daemon cleanly by sending it a TERM or INT signal.

See also

[mod_disk_cache](#)



Synopsis

```
htcacheclean [ -D ] [ -v ] [ -t ] [ -r ] [ -n ] -ppath  
-llimit
```

```
htcacheclean [ -n ] [ -t ] [ -i ] -dinterval -ppath -  
llimit
```



-dinterval

Daemonize and repeat cache cleaning every *interval* minutes. This option is mutually exclusive with the -D, -v and -r options. To shutdown the daemon cleanly, just send it a SIGTERM or SIGINT.

-D

Do a dry run and don't delete anything. This option is mutually exclusive with the -d option.

-v

Be verbose and print statistics. This option is mutually exclusive with the -d option.

-r

Clean thoroughly. This assumes that the Apache web server is not running (otherwise you may get garbage in the cache). This option is mutually exclusive with the -d option and implies the -t option.

-n

Be nice. This causes slower processing in favour of other processes. `htcacheclean` will sleep from time to time so that (a) the disk IO will be delayed and (b) the kernel can schedule other processes in the meantime.

-t

Delete all empty directories. By default only cache files are removed, however with some configurations the large number of directories created may require attention. If your configuration requires a very large number of directories, to the point that inode or file allocation table exhaustion may become an issue, use of this option is advised.

-ppath

Specify *path* as the root directory of the disk cache. This should be the same value as specified with the `CacheRoot` directive.

-llimit

Specify *limit* as the total disk cache size limit. The value is expressed

in bytes by default (or attaching B to the number). Attach K for Kbytes or M for MBytes.

-i

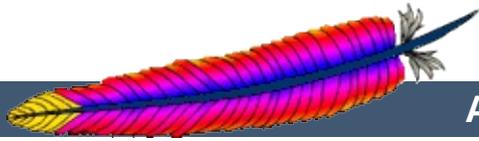
Be intelligent and run only when there was a modification of the disk cache. This option is only possible together with the -d option.



htcacheclean returns a zero status ("true") if all operations were successful, 1 otherwise.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

htdbm - Manipulate DBM password databases

htdbm is used to manipulate the DBM format files used to store usernames and password for basic authentication of HTTP users via [mod_authn_dbm](#). See the [dbmmanage](#) documentation for more information about these DBM files.

See also

[httpd](#)

[dbmmanage](#)

[mod_authn_dbm](#)



Synopsis

```
htdbm [ -TDBTYPE ] [ -c ] [ -m | -d | -p | -s ] [ -t ]  
[ -v ] [ -x ] filename username
```

```
htdbm -b [ -TDBTYPE ] [ -c ] [ -m | -d | -p | -s ] [ -t ]  
[ -v ] filename username password
```

```
htdbm -n [ -c ] [ -m | -d | -p | -s ] [ -t ] [ -v ]  
username
```

```
htdbm -nb [ -c ] [ -m | -d | -p | -s ] [ -t ] [ -v ]  
username password
```

```
htdbm -v [ -TDBTYPE ] [ -c ] [ -m | -d | -p | -s ] [ -t ]  
[ -v ] filename username
```

```
htdbm -vb [ -TDBTYPE ] [ -c ] [ -m | -d | -p | -s ] [ -t ]  
[ -v ] filename username password
```

```
htdbm -x [ -TDBTYPE ] [ -m | -d | -p | -s ] filename username
```

```
htdbm -l [ -TDBTYPE ]
```



-b

Use batch mode; *i.e.*, get the password from the command line rather than prompting for it. This option should be used with extreme care, since **the password is clearly visible** on the command line.

-c

Create the *passwdfile*. If *passwdfile* already exists, it is rewritten and truncated. This option cannot be combined with the `-n` option.

-n

Display the results on standard output rather than updating a database. This option changes the syntax of the command line, since the *passwdfile* argument (usually the first one) is omitted. It cannot be combined with the `-c` option.

-m

Use MD5 encryption for passwords. On Windows, Netware and TPF, this is the default.

-d

Use `crypt()` encryption for passwords. The default on all platforms but Windows, Netware and TPF. Though possibly supported by `htdbm` on all platforms, it is not supported by the [httpd](#) server on Windows, Netware and TPF.

-s

Use SHA encryption for passwords. Facilitates migration from/to Netscape servers using the LDAP Directory Interchange Format (ldif).

-p

Use plaintext passwords. Though `htdbm` will support creation on all platforms, the [httpd](#) daemon will only accept plain text passwords on Windows, Netware and TPF.

-l

Print each of the usernames and comments from the database on

stdout.

-t

Interpret the final parameter as a comment. When this option is specified, an additional string can be appended to the command line; this string will be stored in the "Comment" field of the database, associated with the specified username.

-v

Verify the username and password. The program will print a message indicating whether the supplied password is valid. If the password is invalid, the program exits with error code 3.

-x

Delete user. If the username exists in the specified DBM file, it will be deleted.

filename

The filename of the DBM format file. Usually without the extension .db, .pag, or .dir. If -c is given, the DBM file is created if it does not already exist, or updated if it does exist.

username

The username to create or update in *passwdfile*. If *username* does not exist in this file, an entry is added. If it does exist, the password is changed.

password

The plaintext password to be encrypted and stored in the DBM file. Used only with the -b flag.

-TDBTYPE

Type of DBM file (SDBM, GDBM, DB, or "default").



One should be aware that there are a number of different DBM file formats in existence, and with all likelihood, libraries for more than one format may exist on your system. The three primary examples are SDBM, NDBM, GNU GDBM, and Berkeley/Sleepycat DB 2/3/4. Unfortunately, all these libraries use different file formats, and you must make sure that the file format used by *filename* is the same format that `htdbm` expects to see. `htdbm` currently has no way of determining what type of DBM file it is looking at. If used against the wrong format, will simply return nothing, or may create a different DBM file with a different name, or at worst, it may corrupt the DBM file if you were attempting to write to it.

One can usually use the `file` program supplied with most Unix systems to see what format a DBM file is in.



htdbm returns a zero status ("true") if the username and password have been successfully added or updated in the DBM File. htdbm returns 1 if it encounters some problem accessing files, 2 if there was a syntax problem with the command line, 3 if the password was entered interactively and the verification entry didn't match, 4 if its operation was interrupted, 5 if a value is too long (username, filename, password, or final computed record), 6 if the username contains illegal characters (see the [Restrictions section](#)), and 7 if the file is not a valid DBM password file.



```
htdbm /usr/local/etc/apache/.htdbm-users jsmith
```

Adds or modifies the password for user `jsmith`. The user is prompted for the password. If executed on a Windows system, the password will be encrypted using the modified Apache MD5 algorithm; otherwise, the system's `crypt ()` routine will be used. If the file does not exist, `htdbm` will do nothing except return an error.

```
htdbm -c /home/does/public_html/.htdbm jane
```

Creates a new file and stores a record in it for user `jane`. The user is prompted for the password. If the file exists and cannot be read, or cannot be written, it is not altered and `htdbm` will display a message and return an error status.

```
htdbm -mb /usr/web/.htdbm-all jones Pwd4Steve
```

Encrypts the password from the command line (`Pwd4Steve`) using the MD5 algorithm, and stores it in the specified file.



Web password files such as those managed by `htdbm` should *not* be within the Web server's URI space -- that is, they should not be fetchable with a browser.

The use of the `-b` option is discouraged, since when it is used the unencrypted password appears on the command line.



Restrictions

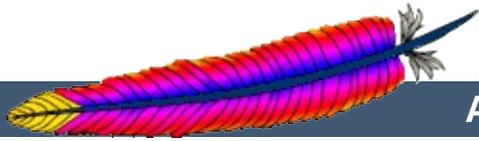
On the Windows and MPE platforms, passwords encrypted with `htdbm` are limited to no more than 255 characters in length. Longer passwords will be truncated to 255 characters.

The MD5 algorithm used by `htdbm` is specific to the Apache software; passwords encrypted using it will not be usable with other Web servers.

Usernames are limited to 255 bytes and may not include the character `:`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

htdigest - manage user files for digest authentication

htdigest is used to create and update the flat-files used to store usernames, realm and password for digest authentication of HTTP users. Resources available from the Apache HTTP server can be restricted to just the users listed in the files created by htdigest.

This manual page only lists the command line arguments. For details of the directives necessary to configure digest authentication in [httpd](#) see the Apache manual, which is part of the Apache distribution or can be found at <http://httpd.apache.org/>.

See also

[httpd](#)

[mod_auth_digest](#)



Synopsis

```
htdigest [ -c ] passwdfile realm username
```



-c

Create the *passwdfile*. If *passwdfile* already exists, it is deleted first.

passwdfile

Name of the file to contain the username, realm and password. If -c is given, this file is created if it does not already exist, or deleted and recreated if it does exist.

realm

The realm name to which the user name belongs. See <http://tools.ietf.org/html/rfc2617#section-3.2.1> for more details.

username

The user name to create or update in *passwdfile*. If *username* does not exist in this file, an entry is added. If it does exist, the password is changed.

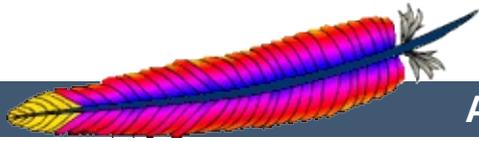


Security Considerations

This program is not safe as a setuid executable. Do *not* make it setuid.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

htpasswd - Manage user files for basic authentication

htpasswd is used to create and update the flat-files used to store usernames and password for basic authentication of HTTP users. If htpasswd cannot access a file, such as not being able to write to the output file or not being able to read the file in order to update it, it returns an error status and makes no changes.

Resources available from the Apache HTTP server can be restricted to just the users listed in the files created by htpasswd. This program can only manage usernames and passwords stored in a flat-file. It can encrypt and display password information for use in other types of data stores, though. To use a DBM database see [dbmmanage](#).

htpasswd encrypts passwords using either a version of MD5 modified for Apache, or the system's crypt () routine. Files managed by htpasswd may contain both types of passwords; some user records may have MD5-encrypted passwords while others in the same file may have passwords encrypted with crypt ().

This manual page only lists the command line arguments. For details of the directives necessary to configure user authentication in [httpd](#) see the Apache manual, which is part of the Apache distribution or can be found at <http://httpd.apache.org/>.

See also

[httpd](#)

The scripts in support/SHA1 which come with the distribution.



Synopsis

htpasswd [**-c**] [**-m**] [**-D**] *passwdfile* *username*

htpasswd -b [**-c**] [**-m** | **-d** | **-p** | **-s**] [**-D**]
passwdfile *username* *password*

htpasswd -n [**-m** | **-d** | **-s** | **-p**] *username*

htpasswd -nb [**-m** | **-d** | **-s** | **-p**] *username* *password*



-b

Use batch mode; *i.e.*, get the password from the command line rather than prompting for it. This option should be used with extreme care, since **the password is clearly visible** on the command line.

-c

Create the *passwdfile*. If *passwdfile* already exists, it is rewritten and truncated. This option cannot be combined with the `-n` option.

-n

Display the results on standard output rather than updating a file. This is useful for generating password records acceptable to Apache for inclusion in non-text data stores. This option changes the syntax of the command line, since the *passwdfile* argument (usually the first one) is omitted. It cannot be combined with the `-c` option.

-m

Use MD5 encryption for passwords. This is the default (since version 2.2.18).

-d

Use `crypt()` encryption for passwords. This is not supported by the [httpd](#) server on Windows and Netware and TPF. This algorithm limits the password length to 8 characters. This algorithm is **insecure** by today's standards. It used to be the default algorithm until version 2.2.17.

-s

Use SHA encryption for passwords. Facilitates migration from/to Netscape servers using the LDAP Directory Interchange Format (ldif).

-p

Use plaintext passwords. Though `htpasswd` will support creation on all platforms, the [httpd](#) daemon will only accept plain text passwords on Windows, Netware and TPF.

-D

Delete user. If the username exists in the specified htpasswd file, it will be deleted.

passwdfile

Name of the file to contain the user name and password. If -c is given, this file is created if it does not already exist, or rewritten and truncated if it does exist.

username

The username to create or update in *passwdfile*. If *username* does not exist in this file, an entry is added. If it does exist, the password is changed.

password

The plaintext password to be encrypted and stored in the file. Only used with the -b flag.



EXIT STATUS

htpasswd returns a zero status ("true") if the username and password have been successfully added or updated in the *passwdfile*. htpasswd returns 1 if it encounters some problem accessing files, 2 if there was a syntax problem with the command line, 3 if the password was entered interactively and the verification entry didn't match, 4 if its operation was interrupted, 5 if a value is too long (username, filename, password, or final computed record), 6 if the username contains illegal characters (see the [Restrictions section](#)), and 7 if the file is not a valid password file.



```
htpasswd /usr/local/etc/apache/.htpasswd-users jsmith
```

Adds or modifies the password for user `jsmith`. The user is prompted for the password. The password will be encrypted using the modified Apache MD5 algorithm. If the file does not exist, `htpasswd` will do nothing except return an error.

```
htpasswd -c /home/does/public_html/.htpasswd jane
```

Creates a new file and stores a record in it for user `jane`. The user is prompted for the password. If the file exists and cannot be read, or cannot be written, it is not altered and `htpasswd` will display a message and return an error status.

```
htpasswd -db /usr/web/.htpasswd-all jones Pwd4Steve
```

Encrypts the password from the command line (`Pwd4Steve`) using the `crypt ()` algorithm, and stores it in the specified file.



Security Considerations

Web password files such as those managed by `htpasswd` should *not* be within the Web server's URI space -- that is, they should not be fetchable with a browser.

This program is not safe as a `setuid` executable. Do *not* make it `setuid`.

The use of the `-b` option is discouraged, since when it is used the unencrypted password appears on the command line.

When using the `crypt ()` algorithm, note that only the first 8 characters of the password are used to form the password. If the supplied password is longer, the extra characters will be silently discarded.

The SHA encryption format does not use salting: for a given password, there is only one encrypted representation. The `crypt ()` and MD5 formats permute the representation by prepending a random salt string, to make dictionary attacks against the passwords more difficult.



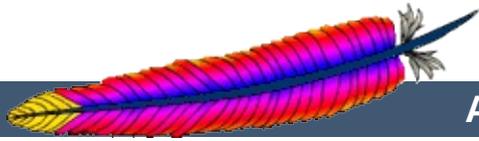
On the Windows and MPE platforms, passwords encrypted with `htpasswd` are limited to no more than 255 characters in length. Longer passwords will be truncated to 255 characters.

The MD5 algorithm used by `htpasswd` is specific to the Apache software; passwords encrypted using it will not be usable with other Web servers.

Username are limited to 255 bytes and may not include the character `:`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

httxt2dbm - Generate dbm files for use with RewriteMap

httxt2dbm is used to generate dbm files from text input, for use in [RewriteMap](#) with the dbm map type.

See also

[httpd](#)

[mod_rewrite](#)



Synopsis

```
httxt2dbm [ -v ] [ -f DBM_TYPE ] -i SOURCE_TXT -o  
OUTPUT_DBM
```



-v

More verbose output

-f *DBM_TYPE*

Specify the DBM type to be used for the output. If not specified, will use the [APR](#) Default. Available types are: GDBM for GDBM files, SDBM for SDBM files, DB for Berkeley DB files, NDBM for NDBM files, default for the default DBM type.

-i *SOURCE_TXT*

Input file from which the dbm is to be created. The file should be formatted with one record per line, of the form: key value. See the documentation for [RewriteMap](#) for further details of this file's format and meaning.

-o *OUTPUT_DBM*

Name of the output dbm files.

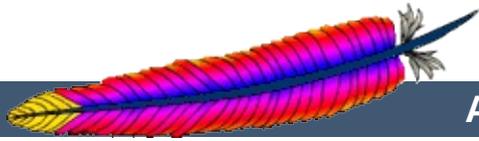


Examples

```
httxt2dbm -i rewritermap.txt -o rewritermap.dbm  
httxt2dbm -f SDBM -i rewritermap.txt -o rewritermap.dbm
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

logresolve - Resolve IP-addresses to hostnames in Apache log files

logresolve is a post-processing program to resolve IP-addresses in Apache's access logfiles. To minimize impact on your nameserver, logresolve has its very own internal hash-table cache. This means that each IP number will only be looked up the first time it is found in the log file.

Takes an Apache log file on standard input. The IP addresses must be the first thing on each line and must be separated from the remainder of the line by a space.



Synopsis

```
logresolve [ -s filename ] [ -c ] < access_log >  
access_log.new
```



-s *filename*

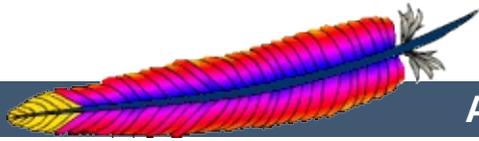
Specifies a filename to record statistics.

-c

This causes `logresolve` to apply some DNS checks: after finding the hostname from the IP address, it looks up the IP addresses for the hostname and checks that one of these matches the original address.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

rotatelogs - Piped logging program to rotate Apache logs

`rotatelog`s is a simple program for use in conjunction with Apache's piped logfile feature. It supports rotation based on a time interval or maximum size of the log.



Synopsis

```
rotatelogs [ -l ] [ -f ] logfile  
rotationtime|filesizeM [ offset ]
```



-l

Causes the use of local time rather than GMT as the base for the interval or for `strftime(3)` formatting with size-based rotation. Note that using `-l` in an environment which changes the GMT offset (such as for BST or DST) can lead to unpredictable results!

-f

Causes the logfile to be opened immediately, as soon as `rotatelogs` starts, instead of waiting for the first logfile entry to be read (for non-busy sites, there may be a substantial delay between when the server is started and when the first request is handled, meaning that the associated logfile does not "exist" until then, which causes problems from some automated logging tools). *Available in version 2.2.9 and later.*

logfile

The path plus basename of the logfile. If *logfile* includes any '%' characters, it is treated as a format string for `strftime(3)`. Otherwise, the suffix `.nnnnnnnnnn` is automatically added and is the time in seconds. Both formats compute the start time from the beginning of the current period. For example, if a rotation time of 86400 is specified, the hour, minute, and second fields created from the `strftime(3)` format will all be zero, referring to the beginning of the current 24-hour period (midnight).

When using `strftime(3)` filename formatting, be sure the log file format has enough granularity to produce a different file name each time the logs are rotated. Otherwise rotation will overwrite the same file instead of starting a new one. For example, if *logfile* was `/var/logs/errorlog.%Y-%m-%d` with log rotation at 5 megabytes, but 5 megabytes was reached twice in the same day, the same log file name would be produced and log rotation would keep writing to the same file.

rotationtime

The time between log file rotations in seconds. The rotation occurs at the beginning of this interval. For example, if the rotation time is 3600, the log file will be rotated at the beginning of every hour; if the rotation time is 86400, the log file will be rotated every night at midnight. (If no data is logged during an interval, no file will be created.)

filesizeM

The maximum file size in megabytes followed by the letter M to specify size rather than time.

offset

The number of minutes offset from UTC. If omitted, zero is assumed and UTC is used. For example, to use local time in the zone UTC -5 hours, specify a value of -300 for this argument. In most cases, -1 should be used instead of specifying an offset.



```
CustomLog "|bin/rotatelogs /var/logs/logfile 86400" common
```

This creates the files `/var/logs/logfile.nnnn` where `nnnn` is the system time at which the log nominally starts (this time will always be a multiple of the rotation time, so you can synchronize cron scripts with it). At the end of each rotation time (here after 24 hours) a new log is started.

```
CustomLog "|bin/rotatelogs -l /var/logs/logfile.%Y.%m.%d 86400" common
```

This creates the files `/var/logs/logfile.yyyy.mm.dd` where `yyyy` is the year, `mm` is the month, and `dd` is the day of the month. Logging will switch to a new file every day at midnight, local time.

```
CustomLog "|bin/rotatelogs /var/logs/logfile 5M" common
```

This configuration will rotate the logfile whenever it reaches a size of 5 megabytes.

```
ErrorLog "|bin/rotatelogs /var/logs/errorlog.%Y-%m-%d-%H_%M_%S 5M"
```

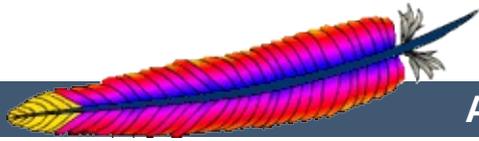
This configuration will rotate the error logfile whenever it reaches a size of 5 megabytes, and the suffix to the logfile name will be created of the form `errorlog.YYYY-mm-dd-HH_MM_SS`.



The following logfile format string substitutions should be supported by all `strptime(3)` implementations, see the `strptime(3)` man page for library-specific extensions.

%A	full weekday name (localized)
%a	3-character weekday name (localized)
%B	full month name (localized)
%b	3-character month name (localized)
%c	date and time (localized)
%d	2-digit day of month
%H	2-digit hour (24 hour clock)
%I	2-digit hour (12 hour clock)
%j	3-digit day of year
%M	2-digit minute
%m	2-digit month
%p	am/pm of 12 hour clock (localized)
%S	2-digit second
%U	2-digit week of year (Sunday first day of week)
%W	2-digit week of year (Monday first day of week)
%w	1-digit weekday (Sunday first day of week)
%X	time (localized)
%x	date (localized)
%Y	4-digit year
%y	2-digit year
%Z	time zone name
%%	literal `%'

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

Other Programs

The following programs are simple support programs included with the Apache HTTP Server which do not have their own manual pages. They are not installed automatically. You can find them after the configuration process in the `support/` directory.



This perl script is designed to be run at a frequent interval by something like cron. It connects to the server and downloads the status information. It reformats the information to a single line and logs it to a file. Adjust the variables at the top of the script to specify the location of the resulting logfile.



split-logfile

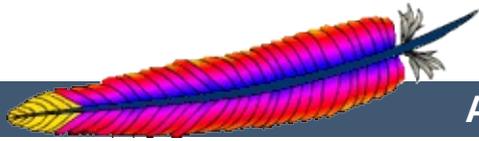
This perl script will take a combined Web server access log file and break its contents into separate files. It assumes that the first field of each line is the virtual host identity (put there by "%v"), and that the logfiles should be named that + ".log" in the current directory.

The combined log file is read from stdin. Records read will be appended to any existing log files.

```
split-logfile < access_log
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Miscellaneous Documentation](#)

Apache Performance Tuning

Apache 2.x is a general-purpose webserver, designed to provide a balance of flexibility, portability, and performance. Although it has not been designed specifically to set benchmark records, Apache 2.x is capable of high performance in many real-world situations.

Compared to Apache 1.3, release 2.x contains many additional optimizations to increase throughput and scalability. Most of these improvements are enabled by default. However, there are compile-time and run-time configuration choices that can significantly affect performance. This document describes the options that a server administrator can configure to tune the performance of an Apache 2.x installation. Some of these configuration options enable the httpd to better take advantage of the capabilities of the hardware and OS, while others allow the administrator to trade functionality for speed.



The single biggest hardware issue affecting webserver performance is RAM. A webserver should never ever have to swap, as swapping increases the latency of each request beyond a point that users consider "fast enough". This causes users to hit stop and reload, further increasing the load. You can, and should, control the `MaxClients` setting so that your server does not spawn so many children that it starts swapping. The procedure for doing this is simple: determine the size of your average Apache process, by looking at your process list via a tool such as `top`, and divide this into your total available memory, leaving some room for other processes.

Beyond that the rest is mundane: get a fast enough CPU, a fast enough network card, and fast enough disks, where "fast enough" is something that needs to be determined by experimentation.

Operating system choice is largely a matter of local concerns. But some guidelines that have proven generally useful are:

- Run the latest stable release and patch level of the operating system that you choose. Many OS suppliers have introduced significant performance improvements to their TCP stacks and thread libraries in recent years.
- If your OS supports a `sendfile(2)` system call, make sure you install the release and/or patches needed to enable it. (With Linux, for example, this means using Linux 2.4 or later. For early releases of Solaris 8, you may need to apply a patch.) On systems where it is available, `sendfile` enables Apache 2 to deliver static content faster and with lower CPU utilization.



Related Modules	Related Directives
mod_dir	AllowOverride
mpm_common	DirectoryIndex
mod_status	HostnameLookups
	EnableMMAP
	EnableSendfile
	KeepAliveTimeout
	MaxSpareServers
	MinSpareServers
	Options
	StartServers

HostnameLookups and other DNS considerations

Prior to Apache 1.3, [HostnameLookups](#) defaulted to On. This adds latency to every request because it requires a DNS lookup to complete before the request is finished. In Apache 1.3 this setting defaults to Off. If you need to have addresses in your log files resolved to hostnames, use the [logresolve](#) program that comes with Apache, or one of the numerous log reporting packages which are available.

It is recommended that you do this sort of postprocessing of your log files on some machine other than the production web server machine, in order that this activity not adversely affect server performance.

If you use any [Allow](#) from domain or [Deny](#) from domain directives (i.e., using a hostname, or a domain name, rather than an IP address) then you will pay for two DNS lookups (a reverse, followed by a forward lookup to make sure that the reverse is not being spoofed). For best performance, therefore, use IP addresses, rather than names, when using these directives, if possible.

Note that it's possible to scope the directives, such as within a

<Location /server-status> section. In this case the DNS lookups are only performed on requests matching the criteria. Here's an example which disables lookups except for .html and .cgi files:

```
HostnameLookups off
<Files ~ "\.(html|cgi)$">
  HostnameLookups on
</Files>
```

But even still, if you just need DNS names in some CGIs you could consider doing the `gethostbyname` call in the specific CGIs that need it.

FollowSymLinks and SymLinksIfOwnerMatch

Wherever in your URL-space you do not have an `Options FollowSymLinks`, or you do have an `Options SymLinksIfOwnerMatch`, Apache will need to issue extra system calls to check up on symlinks. (One extra call per filename component.) For example, if you had:

```
DocumentRoot /www/htdocs
<Directory />
  Options SymLinksIfOwnerMatch
</Directory>
```

and a request is made for the URI `/index.html`, then Apache will perform `lstat(2)` on `/www`, `/www/htdocs`, and `/www/htdocs/index.html`. The results of these `lstats` are never cached, so they will occur on every single request. If you really desire the symlinks security checking, you can do something like this:

```
DocumentRoot /www/htdocs
<Directory />
  Options FollowSymLinks
</Directory>

<Directory /www/htdocs>
  Options -FollowSymLinks +SymLinksIfOwnerMatch
```

```
</Directory>
```

This at least avoids the extra checks for the [DocumentRoot](#) path. Note that you'll need to add similar sections if you have any [Alias](#) or [RewriteRule](#) paths outside of your document root. For highest performance, and no symlink protection, set `FollowSymLinks` everywhere, and never set `SymLinksIfOwnerMatch`.

AllowOverride

Wherever in your URL-space you allow overrides (typically `.htaccess` files), Apache will attempt to open `.htaccess` for each filename component. For example,

```
DocumentRoot /www/htdocs
<Directory />
    AllowOverride all
</Directory>
```

and a request is made for the URI `/index.html`. Then Apache will attempt to open `/.htaccess`, `/www/.htaccess`, and `/www/htdocs/.htaccess`. The solutions are similar to the previous case of `Options FollowSymLinks`. For highest performance use `AllowOverride None` everywhere in your filesystem.

Negotiation

If at all possible, avoid content negotiation if you're really interested in every last ounce of performance. In practice the benefits of negotiation outweigh the performance penalties. There's one case where you can speed up the server. Instead of using a wildcard such as:

```
DirectoryIndex index
```

Use a complete list of options:

```
DirectoryIndex index.cgi index.pl index.shtml index.html
```

where you list the most common choice first.

Also note that explicitly creating a type-map file provides better performance than using `MultiViews`, as the necessary information can be determined by reading this single file, rather than having to scan the directory for files.

If your site needs content negotiation, consider using type-map files, rather than the `Options MultiViews` directive to accomplish the negotiation. See the [Content Negotiation](#) documentation for a full discussion of the methods of negotiation, and instructions for creating type-map files.

Memory-mapping

In situations where Apache 2.x needs to look at the contents of a file being delivered--for example, when doing server-side-include processing--it normally memory-maps the file if the OS supports some form of `mmap(2)`.

On some platforms, this memory-mapping improves performance. However, there are cases where memory-mapping can hurt the performance or even the stability of the `httpd`:

- On some operating systems, `mmap` does not scale as well as `read(2)` when the number of CPUs increases. On multiprocessor Solaris servers, for example, Apache 2.x sometimes delivers server-parsed files faster when `mmap` is disabled.
- If you memory-map a file located on an NFS-mounted filesystem and a process on another NFS client machine deletes or truncates the file, your process may get a bus error the next time it tries to access the mapped file content.

For installations where either of these factors applies, you should use `EnableMMAP off` to disable the memory-mapping of delivered files. (Note: This directive can be overridden on a per-directory basis.)

Sendfile

In situations where Apache 2.x can ignore the contents of the file to be delivered -- for example, when serving static file content -- it normally uses the kernel sendfile support for the file if the OS supports the `sendfile(2)` operation.

On most platforms, using sendfile improves performance by eliminating separate read and send mechanics. However, there are cases where using sendfile can harm the stability of the httpd:

- Some platforms may have broken sendfile support that the build system did not detect, especially if the binaries were built on another box and moved to such a machine with broken sendfile support.
- With an NFS-mounted files, the kernel may be unable to reliably serve the network file through it's own cache.

For installations where either of these factors applies, you should use `EnableSendfile off` to disable sendfile delivery of file contents. (Note: This directive can be overridden on a per-directory basis.)

Process Creation

Prior to Apache 1.3 the [MinSpareServers](#), [MaxSpareServers](#), and [StartServers](#) settings all had drastic effects on benchmark results. In particular, Apache required a "ramp-up" period in order to reach a number of children sufficient to serve the load being applied. After the initial spawning of [StartServers](#) children, only one child per second would be created to satisfy the [MinSpareServers](#) setting. So a server being accessed by 100 simultaneous clients, using the default

[StartServers](#) of 5 would take on the order of 95 seconds to spawn enough children to handle the load. This works fine in practice on real-life servers because they aren't restarted frequently. But it does really poorly on benchmarks which might only run for ten minutes.

The one-per-second rule was implemented in an effort to avoid swamping the machine with the startup of new children. If the machine is busy spawning children, it can't service requests. But it has such a drastic effect on the perceived performance of Apache that it had to be replaced. As of Apache 1.3, the code will relax the one-per-second rule. It will spawn one, wait a second, then spawn two, wait a second, then spawn four, and it will continue exponentially until it is spawning 32 children per second. It will stop whenever it satisfies the [MinSpareServers](#) setting.

This appears to be responsive enough that it's almost unnecessary to twiddle the [MinSpareServers](#), [MaxSpareServers](#) and [StartServers](#) knobs. When more than 4 children are spawned per second, a message will be emitted to the [ErrorLog](#). If you see a lot of these errors, then consider tuning these settings. Use the [mod_status](#) output as a guide.

Related to process creation is process death induced by the [MaxRequestsPerChild](#) setting. By default this is 0, which means that there is no limit to the number of requests handled per child. If your configuration currently has this set to some very low number, such as 30, you may want to bump this up significantly. If you are running SunOS or an old version of Solaris, limit this to 10000 or so because of memory leaks.

When keep-alives are in use, children will be kept busy doing nothing waiting for more requests on the already open connection. The default [KeepAliveTimeout](#) of 5 seconds attempts to minimize this effect. The tradeoff here is between network bandwidth and server resources. In no

event should you raise this above about 60 seconds, as [most of the benefits are lost](#).



Choosing an MPM

Apache 2.x supports pluggable concurrency models, called [Multi-Processing Modules](#) (MPMs). When building Apache, you must choose an MPM to use. There are platform-specific MPMs for some platforms: [beos](#), [mpm_netware](#), [mpmt_os2](#), and [mpm_winnt](#). For general Unix-type systems, there are several MPMs from which to choose. The choice of MPM can affect the speed and scalability of the httpd:

- The [worker](#) MPM uses multiple child processes with many threads each. Each thread handles one connection at a time. Worker generally is a good choice for high-traffic servers because it has a smaller memory footprint than the prefork MPM.
- The [prefork](#) MPM uses multiple child processes with one thread each. Each process handles one connection at a time. On many systems, prefork is comparable in speed to worker, but it uses more memory. Prefork's threadless design has advantages over worker in some situations: it can be used with non-thread-safe third-party modules, and it is easier to debug on platforms with poor thread debugging support.

For more information on these and other MPMs, please see the [MPM documentation](#).

Modules

Since memory usage is such an important consideration in performance, you should attempt to eliminate modules that you are not actually using. If you have built the modules as [DSOs](#), eliminating modules is a simple matter of commenting out the associated [LoadModule](#) directive for that module. This allows you to experiment with removing modules and seeing if your site still functions in their absence.

If, on the other hand, you have modules statically linked into your Apache

binary, you will need to recompile Apache in order to remove unwanted modules.

An associated question that arises here is, of course, what modules you need, and which ones you don't. The answer here will, of course, vary from one web site to another. However, the *minimal* list of modules which you can get by with tends to include [mod_mime](#), [mod_dir](#), and [mod_log_config](#). `mod_log_config` is, of course, optional, as you can run a web site without log files. This is, however, not recommended.

Atomic Operations

Some modules, such as [mod_cache](#) and recent development builds of the worker MPM, use APR's atomic API. This API provides atomic operations that can be used for lightweight thread synchronization.

By default, APR implements these operations using the most efficient mechanism available on each target OS/CPU platform. Many modern CPUs, for example, have an instruction that does an atomic compare-and-swap (CAS) operation in hardware. On some platforms, however, APR defaults to a slower, mutex-based implementation of the atomic API in order to ensure compatibility with older CPU models that lack such instructions. If you are building Apache for one of these platforms, and you plan to run only on newer CPUs, you can select a faster atomic implementation at build time by configuring Apache with the `--enable-nonportable-atomics` option:

```
./buildconf
./configure --with-mpm=worker --enable-nonportable-atomics=yes
```

The `--enable-nonportable-atomics` option is relevant for the following platforms:

- Solaris on SPARC
By default, APR uses mutex-based atomics on Solaris/SPARC. If

you configure with `--enable-nonportable-atomics`, however, APR generates code that uses a SPARC v8plus opcode for fast hardware compare-and-swap. If you configure Apache with this option, the atomic operations will be more efficient (allowing for lower CPU utilization and higher concurrency), but the resulting executable will run only on UltraSPARC chips.

- Linux on x86
By default, APR uses mutex-based atomics on Linux. If you configure with `--enable-nonportable-atomics`, however, APR generates code that uses a 486 opcode for fast hardware compare-and-swap. This will result in more efficient atomic operations, but the resulting executable will run only on 486 and later chips (and not on 386).

mod_status and ExtendedStatus On

If you include `mod_status` and you also set `ExtendedStatus On` when building and running Apache, then on every request Apache will perform two calls to `gettimeofday(2)` (or `times(2)` depending on your operating system), and (pre-1.3) several extra calls to `time(2)`. This is all done so that the status report contains timing indications. For highest performance, set `ExtendedStatus off` (which is the default).

accept Serialization - Multiple Sockets

Warning:

This section has not been fully updated to take into account changes made in the 2.x version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

This discusses a shortcoming in the Unix socket API. Suppose your web server uses multiple `Listen` statements to listen on either multiple ports or multiple addresses. In order to test each socket to see if a connection

is ready, Apache uses `select(2)`. `select(2)` indicates that a socket has *zero* or *at least one* connection waiting on it. Apache's model includes multiple children, and all the idle ones test for new connections at the same time. A naive implementation looks something like this (these examples do not match the code, they're contrived for pedagogical purposes):

```
for (;;) {
    for (;;) {
        fd_set accept_fds;

        FD_ZERO (&accept_fds);
        for (i = first_socket; i <= last_socket; ++i) {
            FD_SET (i, &accept_fds);
        }
        rc = select (last_socket+1, &accept_fds, NULL, NULL, NULL);
        if (rc < 1) continue;
        new_connection = -1;
        for (i = first_socket; i <= last_socket; ++i) {
            if (FD_ISSET (i, &accept_fds)) {
                new_connection = accept (i, NULL, NULL);
                if (new_connection != -1) break;
            }
        }
        if (new_connection != -1) break;
    }
    process the new_connection;
}
```

But this naive implementation has a serious starvation problem. Recall that multiple children execute this loop at the same time, and so multiple children will block at `select` when they are in between requests. All those blocked children will awaken and return from `select` when a single request appears on any socket. (The number of children which awaken varies depending on the operating system and timing issues.) They will all then fall down into the loop and try to accept the connection. But only one will succeed (assuming there's still only one connection ready). The rest will be *blocked* in `accept`. This effectively locks those children into serving requests from that one socket and no other sockets, and they'll be stuck there until enough new requests

appear on that socket to wake them all up. This starvation problem was first documented in [PR#467](#). There are at least two solutions.

One solution is to make the sockets non-blocking. In this case the `accept` won't block the children, and they will be allowed to continue immediately. But this wastes CPU time. Suppose you have ten idle children in `select`, and one connection arrives. Then nine of those children will wake up, try to accept the connection, fail, and loop back into `select`, accomplishing nothing. Meanwhile none of those children are servicing requests that occurred on other sockets until they get back up to the `select` again. Overall this solution does not seem very fruitful unless you have as many idle CPUs (in a multiprocessor box) as you have idle children (not a very likely situation).

Another solution, the one used by Apache, is to serialize entry into the inner loop. The loop looks like this (differences highlighted):

```
for (;;) {
    accept_mutex_on ();
    for (;;) {
        fd_set accept_fds;

        FD_ZERO (&accept_fds);
        for (i = first_socket; i <= last_socket; ++i) {
            FD_SET (i, &accept_fds);
        }
        rc = select (last_socket+1, &accept_fds, NULL, NULL, NULL);
        if (rc < 1) continue;
        new_connection = -1;
        for (i = first_socket; i <= last_socket; ++i) {
            if (FD_ISSET (i, &accept_fds)) {
                new_connection = accept (i, NULL, NULL);
                if (new_connection != -1) break;
            }
        }
        if (new_connection != -1) break;
    }
    accept_mutex_off ();
    process the new_connection;
}
```

The functions `accept_mutex_on` and `accept_mutex_off` implement a mutual exclusion semaphore. Only one child can have the mutex at any time. There are several choices for implementing these mutexes. The choice is defined in `src/conf.h` (pre-1.3) or `src/include/ap_config.h` (1.3 or later). Some architectures do not have any locking choice made, on these architectures it is unsafe to use multiple [Listen](#) directives.

The directive [AcceptMutex](#) can be used to change the selected mutex implementation at run-time.

AcceptMutex flock

This method uses the `flock(2)` system call to lock a lock file (located by the [LockFile](#) directive).

AcceptMutex fcntl

This method uses the `fcntl(2)` system call to lock a lock file (located by the [LockFile](#) directive).

AcceptMutex sysvsem

(1.3 or later) This method uses SysV-style semaphores to implement the mutex. Unfortunately SysV-style semaphores have some bad side-effects. One is that it's possible Apache will die without cleaning up the semaphore (see the `ipc(8)` man page). The other is that the semaphore API allows for a denial of service attack by any CGIs running under the same uid as the webserver (*i.e.*, all CGIs, unless you use something like [suexec](#) or `cgiwrapper`). For these reasons this method is not used on any architecture except IRIX (where the previous two are prohibitively expensive on most IRIX boxes).

AcceptMutex pthread

(1.3 or later) This method uses POSIX mutexes and should work on any architecture implementing the full POSIX threads specification, however appears to only work on Solaris (2.5 or later), and even

then only in certain configurations. If you experiment with this you should watch out for your server hanging and not responding. Static content only servers may work just fine.

AcceptMutex posixsem

(2.0 or later) This method uses POSIX semaphores. The semaphore ownership is not recovered if a thread in the process holding the mutex segfaults, resulting in a hang of the web server.

If your system has another method of serialization which isn't in the above list then it may be worthwhile adding code for it to APR.

Another solution that has been considered but never implemented is to partially serialize the loop -- that is, let in a certain number of processes. This would only be of interest on multiprocessor boxes where it's possible that multiple children could run simultaneously, and the serialization actually doesn't take advantage of the full bandwidth. This is a possible area of future investigation, but priority remains low because highly parallel web servers are not the norm.

Ideally you should run servers without multiple [Listen](#) statements if you want the highest performance. But read on.

accept Serialization - Single Socket

The above is fine and dandy for multiple socket servers, but what about single socket servers? In theory they shouldn't experience any of these same problems because all children can just block in `accept (2)` until a connection arrives, and no starvation results. In practice this hides almost the same "spinning" behavior discussed above in the non-blocking solution. The way that most TCP stacks are implemented, the kernel actually wakes up all processes blocked in `accept` when a single connection arrives. One of those processes gets the connection and returns to user-space. The rest spin in the kernel and go back to sleep when they discover there's no connection for them. This spinning is

hidden from the user-land code, but it's there nonetheless. This can result in the same load-spiking wasteful behavior that a non-blocking solution to the multiple sockets case can.

For this reason we have found that many architectures behave more "nicely" if we serialize even the single socket case. So this is actually the default in almost all cases. Crude experiments under Linux (2.0.30 on a dual Pentium pro 166 w/128Mb RAM) have shown that the serialization of the single socket case causes less than a 3% decrease in requests per second over unserialized single-socket. But unserialized single-socket showed an extra 100ms latency on each request. This latency is probably a wash on long haul lines, and only an issue on LANs. If you want to override the single socket serialization, you can define `SINGLE_LISTEN_UNSERIALIZED_ACCEPT`, and then single-socket servers will not serialize at all.

Lingering Close

As discussed in [draft-ietf-http-connection-00.txt](#) section 8, in order for an HTTP server to **reliably** implement the protocol, it needs to shut down each direction of the communication independently. (Recall that a TCP connection is bi-directional. Each half is independent of the other.) This fact is often overlooked by other servers, but is correctly implemented in Apache as of 1.2.

When this feature was added to Apache, it caused a flurry of problems on various versions of Unix because of shortsightedness. The TCP specification does not state that the `FIN_WAIT_2` state has a timeout, but it doesn't prohibit it. On systems without the timeout, Apache 1.2 induces many sockets stuck forever in the `FIN_WAIT_2` state. In many cases this can be avoided by simply upgrading to the latest TCP/IP patches supplied by the vendor. In cases where the vendor has never released patches (*i.e.*, SunOS4 -- although folks with a source license can patch it themselves), we have decided to disable this feature.

There are two ways to accomplish this. One is the socket option `SO_LINGER`. But as fate would have it, this has never been implemented properly in most TCP/IP stacks. Even on those stacks with a proper implementation (*i.e.*, Linux 2.0.31), this method proves to be more expensive (cputime) than the next solution.

For the most part, Apache implements this in a function called `lingering_close` (in `http_main.c`). The function looks roughly like this:

```
void lingering_close (int s)
{
    char junk_buffer[2048];

    /* shutdown the sending side */
    shutdown (s, 1);

    signal (SIGALRM, lingering_death);
    alarm (30);

    for (;;) {
        select (s for reading, 2 second timeout);
        if (error) break;
        if (s is ready for reading) {
            if (read (s, junk_buffer, sizeof (junk_buffer)) <= 0) {
                break;
            }
            /* just toss away whatever is here */
        }
    }

    close (s);
}
```

This naturally adds some expense at the end of a connection, but it is required for a reliable implementation. As HTTP/1.1 becomes more prevalent, and all connections are persistent, this expense will be amortized over more requests. If you want to play with fire and disable this feature, you can define `NO_LINGCLOSE`, but this is not recommended at all. In particular, as HTTP/1.1 pipelined persistent connections come into use, `lingering_close` is an absolute necessity

(and [pipelined connections are faster](#), so you want to support them).

Scoreboard File

Apache's parent and children communicate with each other through something called the scoreboard. Ideally this should be implemented in shared memory. For those operating systems that we either have access to, or have been given detailed ports for, it typically is implemented using shared memory. The rest default to using an on-disk file. The on-disk file is not only slow, but it is unreliable (and less featured). Peruse the `src/main/conf.h` file for your architecture, and look for either `USE_MMAP_SCOREBOARD` or `USE_SHMGET_SCOREBOARD`. Defining one of those two (as well as their companions `HAVE_MMAP` and `HAVE_SHMGET` respectively) enables the supplied shared memory code. If your system has another type of shared memory, edit the file `src/main/http_main.c` and add the hooks necessary to use it in Apache. (Send us back a patch too, please.)

Historical note: The Linux port of Apache didn't start to use shared memory until version 1.2 of Apache. This oversight resulted in really poor and unreliable behavior of earlier versions of Apache on Linux.

DYNAMIC_MODULE_LIMIT

If you have no intention of using dynamically loaded modules (you probably don't if you're reading this and tuning your server for every last ounce of performance), then you should add - `DDYNAMIC_MODULE_LIMIT=0` when building your server. This will save RAM that's allocated only for supporting dynamically loaded modules.



Here is a system call trace of Apache 2.0.38 with the worker MPM on Solaris 8. This trace was collected using:

```
truss -l -p httpd_child_pid.
```

The `-l` option tells `truss` to log the ID of the LWP (lightweight process--Solaris's form of kernel-level thread) that invokes each system call.

Other systems may have different system call tracing utilities such as `strace`, `ktrace`, or `par`. They all produce similar output.

In this trace, a client has requested a 10KB static file from the `httpd`. Traces of non-static requests or requests with content negotiation look wildly different (and quite ugly in some cases).

```
/67:    accept(3, 0x00200BEC, 0x00200C0C, 1) (sleeping...)
/67:    accept(3, 0x00200BEC, 0x00200C0C, 1)                = 9
```

In this trace, the listener thread is running within LWP #67.

Note the lack of `accept(2)` serialization. On this particular platform, the worker MPM uses an unserialized `accept` by default unless it is listening on multiple ports.

```
/65:    lwp_park(0x00000000, 0)                               = 0
/67:    lwp_unpark(65, 1)                                   = 0
```

Upon accepting the connection, the listener thread wakes up a worker thread to do the request processing. In this trace, the worker thread that handles the request is mapped to LWP #65.

```
/65:    getsockname(9, 0x00200BA4, 0x00200BC4, 1)          = 0
```

In order to implement virtual hosts, Apache needs to know the local

socket address used to accept the connection. It is possible to eliminate this call in many situations (such as when there are no virtual hosts, or when [Listen](#) directives are used which do not have wildcard addresses). But no effort has yet been made to do these optimizations.

```
/65:   brk(0x002170E8)           = 0
/65:   brk(0x002190E8)           = 0
```

The `brk(2)` calls allocate memory from the heap. It is rare to see these in a system call trace, because the `httpd` uses custom memory allocators (`apr_pool` and `apr_bucket_alloc`) for most request processing. In this trace, the `httpd` has just been started, so it must call `malloc(3)` to get the blocks of raw memory with which to create the custom memory allocators.

```
/65:  fcntl(9, F_GETFL, 0x00000000)       = 2
/65:   fstat64(9, 0xFAF7B818)             = 0
/65:   getsockopt(9, 65535, 8192, 0xFAF7B918, 0xFAF7B910, 2190656) = 0
/65:   fstat64(9, 0xFAF7B818)             = 0
/65:   getsockopt(9, 65535, 8192, 0xFAF7B918, 0xFAF7B914, 2190656) = 0
/65:   setsockopt(9, 65535, 8192, 0xFAF7B918, 4, 2190656) = 0
/65:   fcntl(9, F_SETFL, 0x00000082)       = 0
```

Next, the worker thread puts the connection to the client (file descriptor 9) in non-blocking mode. The `setsockopt(2)` and `getsockopt(2)` calls are a side-effect of how Solaris's `libc` handles `fcntl(2)` on sockets.

```
/65:   read(9, " G E T / 1 0 k . h t m"..., 8000)   = 97
```

The worker thread reads the request from the client.

```
/65:   stat("/var/httpd/apache/httpd-8999/htdocs/10k.html", 0xFAF7B978)
/65:   open("/var/httpd/apache/httpd-8999/htdocs/10k.html", O_RDONLY)
```

This `httpd` has been configured with `Options FollowSymLinks` and `AllowOverride None`. Thus it doesn't need to `lstat(2)` each

directory in the path leading up to the requested file, nor check for .htaccess files. It simply calls `stat(2)` to verify that the file: 1) exists, and 2) is a regular file, not a directory.

```
/65:    sendfilev(0, 9, 0x00200F90, 2, 0xFAF7B53C)    = 10269
```

In this example, the `httpd` is able to send the HTTP response header and the requested file with a single `sendfilev(2)` system call. `Sendfile` semantics vary among operating systems. On some other systems, it is necessary to do a `write(2)` or `writew(2)` call to send the headers before calling `sendfile(2)`.

```
/65:    write(4, " 1 2 7 . 0 . 0 . 1 - ..", 78)    = 78
```

This `write(2)` call records the request in the access log. Note that one thing missing from this trace is a `time(2)` call. Unlike Apache 1.3, Apache 2.x uses `gettimeofday(3)` to look up the time. On some operating systems, like Linux or Solaris, `gettimeofday` has an optimized implementation that doesn't require as much overhead as a typical system call.

```
/65:    shutdown(9, 1, 1)    = 0
/65:    poll(0xFAF7B980, 1, 2000)    = 1
/65:    read(9, 0xFAF7BC20, 512)    = 0
/65:    close(9)    = 0
```

The worker thread does a lingering close of the connection.

```
/65:    close(10)    = 0
/65:    lwp_park(0x00000000, 0)    (sleeping...)
```

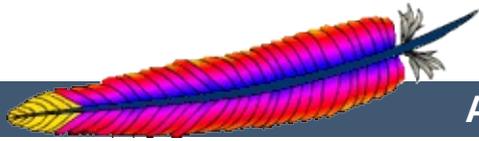
Finally the worker thread closes the file that it has just delivered and blocks until the listener assigns it another connection.

```
/67:    accept(3, 0x001FEB74, 0x001FEB94, 1) (sleeping...)
```

Meanwhile, the listener thread is able to accept another connection as soon as it has dispatched this connection to a worker thread (subject to some flow-control logic in the worker MPM that throttles the listener if all the available workers are busy). Though it isn't apparent from this trace, the next accept (2) can (and usually does, under high load conditions) occur in parallel with the worker thread's handling of the just-accepted connection.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Miscellaneous Documentation](#)

Security Tips

Some hints and tips on security issues in setting up a web server.
Some of the suggestions will be general, others specific to Apache.



The Apache HTTP Server has a good record for security and a developer community highly concerned about security issues. But it is inevitable that some problems -- small or large -- will be discovered in software after it is released. For this reason, it is crucial to keep aware of updates to the software. If you have obtained your version of the HTTP Server directly from Apache, we highly recommend you subscribe to the [Apache HTTP Server Announcements List](#) where you can keep informed of new releases and security updates. Similar services are available from most third-party distributors of Apache software.

Of course, most times that a web server is compromised, it is not because of problems in the HTTP Server code. Rather, it comes from problems in add-on code, CGI scripts, or the underlying Operating System. You must therefore stay aware of problems and updates with all the software on your system.



In typical operation, Apache is started by the root user, and it switches to the user defined by the `User` directive to serve hits. As is the case with any command that root executes, you must take care that it is protected from modification by non-root users. Not only must the files themselves be writeable only by root, but so must the directories, and parents of all directories. For example, if you choose to place `ServerRoot` in `/usr/local/apache` then it is suggested that you create that directory as root, with commands like these:

```
mkdir /usr/local/apache
cd /usr/local/apache
mkdir bin conf logs
chown 0 . bin conf logs
chgrp 0 . bin conf logs
chmod 755 . bin conf logs
```

It is assumed that `/`, `/usr`, and `/usr/local` are only modifiable by root. When you install the `httpd` executable, you should ensure that it is similarly protected:

```
cp httpd /usr/local/apache/bin
chown 0 /usr/local/apache/bin/httpd
chgrp 0 /usr/local/apache/bin/httpd
chmod 511 /usr/local/apache/bin/httpd
```

You can create an `htdocs` subdirectory which is modifiable by other users -- since root never executes any files out of there, and shouldn't be creating files in there.

If you allow non-root users to modify any files that root either executes or writes on then you open your system to root compromises. For example, someone could replace the `httpd` binary so that the next time you start it, it will execute some arbitrary code. If the logs directory is writeable (by a non-root user), someone could replace a log file with a symlink to some other system file, and then root might overwrite that file with arbitrary data. If the log files themselves are writeable (by a non-root user), then

someone may be able to overwrite the log itself with bogus data.



Server Side Includes (SSI) present a server administrator with several potential security risks.

The first risk is the increased load on the server. All SSI-enabled files have to be parsed by Apache, whether or not there are any SSI directives included within the files. While this load increase is minor, in a shared server environment it can become significant.

SSI files also pose the same risks that are associated with CGI scripts in general. Using the `exec` `cmd` element, SSI-enabled files can execute any CGI script or program under the permissions of the user and group Apache runs as, as configured in `httpd.conf`.

There are ways to enhance the security of SSI files while still taking advantage of the benefits they provide.

To isolate the damage a wayward SSI file can cause, a server administrator can enable [suexec](#) as described in the [CGI in General](#) section.

Enabling SSI for files with `.html` or `.htm` extensions can be dangerous. This is especially true in a shared, or high traffic, server environment. SSI-enabled files should have a separate extension, such as the conventional `.shtml`. This helps keep server load at a minimum and allows for easier management of risk.

Another solution is to disable the ability to run scripts and programs from SSI pages. To do this replace `Includes` with `IncludesNOEXEC` in the [Options](#) directive. Note that users may still use `<--#include virtual="..." -->` to execute CGI scripts if these scripts are in directories designated by a [ScriptAlias](#) directive.



First of all, you always have to remember that you must trust the writers of the CGI scripts/programs or your ability to spot potential security holes in CGI, whether they were deliberate or accidental. CGI scripts can run essentially arbitrary commands on your system with the permissions of the web server user and can therefore be extremely dangerous if they are not carefully checked.

All the CGI scripts will run as the same user, so they have potential to conflict (accidentally or deliberately) with other scripts e.g. User A hates User B, so he writes a script to trash User B's CGI database. One program which can be used to allow scripts to run as different users is [suEXEC](#) which is included with Apache as of 1.2 and is called from special hooks in the Apache server code. Another popular way of doing this is with [CGIWrap](#).



Allowing users to execute CGI scripts in any directory should only be considered if:

- You trust your users not to write scripts which will deliberately or accidentally expose your system to an attack.
- You consider security at your site to be so feeble in other areas, as to make one more potential hole irrelevant.
- You have no users, and nobody ever visits your server.



Limiting CGI to special directories gives the admin control over what goes into those directories. This is inevitably more secure than non script aliased CGI, but only if users with write access to the directories are trusted or the admin is willing to test each new CGI script/program for potential security holes.

Most sites choose this option over the non script aliased CGI approach.



Embedded scripting options which run as part of the server itself, such as `mod_php`, `mod_perl`, `mod_tcl`, and `mod_python`, run under the identity of the server itself (see the [User](#) directive), and therefore scripts executed by these engines potentially can access anything the server user can. Some scripting engines may provide restrictions, but it is better to be safe and assume not.



Preventing System Changes

To run a really tight ship, you'll want to stop users from setting up .htaccess files which can override security features you've configured. Here's one way to do it.

In the server configuration file, put

```
<Directory />  
AllowOverride None  
</Directory>
```

This prevents the use of .htaccess files in all directories apart from those specifically enabled.



Prevent Default Access by Denial

One aspect of Apache which is occasionally misunderstood is the feature of default access. That is, unless you take steps to change it, if the server can find its way to a file through normal URL mapping rules, it can serve it to clients.

For instance, consider the following example:

```
# cd /; ln -s / public_html
Accessing http://localhost/~root/
```

This would allow clients to walk through the entire filesystem. To work around this, add the following block to your server's configuration:

```
<Directory />
Order Deny,Allow
Deny from all
</Directory>
```

This will forbid default access to filesystem locations. Add appropriate [Directory](#) blocks to allow access only in those areas you wish. For example,

```
<Directory /usr/users/*/public_html>
Order Deny,Allow
Allow from all
</Directory>
<Directory /usr/local/httpd>
Order Deny,Allow
Allow from all
</Directory>
```

Pay particular attention to the interactions of [Location](#) and [Directory](#) directives; for instance, even if `<Directory />` denies access, a `<Location />` directive might overturn it.

Also be wary of playing games with the [UserDir](#) directive; setting it to something like `./` would have the same effect, for root, as the first

example above. If you are using Apache 1.3 or above, we strongly recommend that you include the following line in your server configuration files:

```
UserDir disabled root
```



Matching Your Logs

To keep up-to-date with what is actually going on against your server you have to check the [Log Files](#). Even though the log files only reports what has already happened, they will give you some understanding of what attacks is thrown against the server and allow you to check if the necessary level of security is present.

A couple of examples:

```
grep -c "/jsp/source.jsp?/jsp/ /jsp/source.jsp??" access_log  
grep "client denied" error_log | tail -n 10
```

The first example will list the number of attacks trying to exploit the [Apache Tomcat Source.JSP Malformed Request Information Disclosure Vulnerability](#), the second example will list the ten last denied clients, for example:

```
[Thu Jul 11 17:18:39 2002] [error] [client foo.example.com] client  
denied by server configuration: /usr/local/apache/htdocs/.htpasswd
```

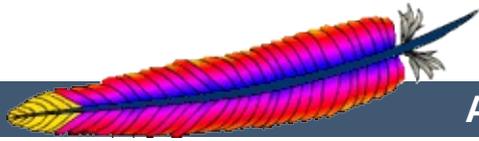
As you can see, the log files only report what already has happened, so if the client had been able to access the `.htpasswd` file you would have seen something similar to:

```
foo.example.com - - [12/Jul/2002:01:59:13 +0200] "GET /.htpasswd  
HTTP/1.1"
```

in your [Access Log](#). This means you probably commented out the following in your server configuration file:

```
<Files ~ "^\.ht">  
Order allow,deny  
Deny from all  
</Files>
```

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Miscellaneous Documentation](#)

Relevant Standards

This page documents all the relevant standards that the Apache HTTP Server follows, along with brief descriptions.

In addition to the information listed below, the following resources should be consulted:

- <http://purl.org/NET/http-errata> - HTTP/1.1 Specification Errata
- <http://www.rfc-editor.org/errata.php> - RFC Errata
- <http://ftp.ics.uci.edu/pub/ietf/http/#RFC> - A pre-compiled list of HTTP related RFCs

Notice

This document is not yet complete.



Regardless of what modules are compiled and used, Apache as a basic web server complies with the following IETF recommendations:

RFC 1945 (Informational)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. This documents HTTP/1.0.

RFC 2616 (Standards Track)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This documents HTTP/1.1.

RFC 2396 (Standards Track)

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource.



Regarding the Hypertext Markup Language, Apache complies with the following IETF and W3C recommendations:

[RFC 2854 \(Informational\)](#)

This document summarizes the history of HTML development, and defines the "text/html" MIME type by pointing to the relevant W3C recommendations.

[HTML 4.01 Specification \(Errata\)](#)

This specification defines the HyperText Markup Language (HTML), the publishing language of the World Wide Web. This specification defines HTML 4.01, which is a subversion of HTML 4.

[HTML 3.2 Reference Specification](#)

The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents.

[XHTML 1.1 - Module-based XHTML \(Errata\)](#)

This Recommendation defines a new XHTML document type that is based upon the module framework and modules defined in Modularization of XHTML.

[XHTML 1.0 The Extensible HyperText Markup Language \(Second Edition\) \(Errata\)](#)

This specification defines the Second Edition of XHTML 1.0, a reformulation of HTML 4 as an XML 1.0 application, and three DTDs corresponding to the ones defined by HTML 4.



Concerning the different methods of authentication, Apache follows the following IETF recommendations:

RFC 2617 (Draft standard)

"HTTP/1.0", includes the specification for a Basic Access Authentication scheme.



The following links document ISO and other language and country code information:

[ISO 639-2](#)

ISO 639 provides two sets of language codes, one as a two-letter code set (639-1) and another as a three-letter code set (this part of ISO 639) for the representation of names of languages.

[ISO 3166-1](#)

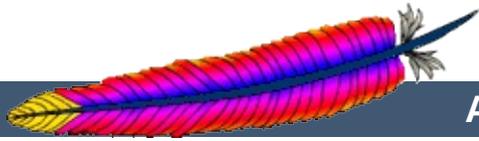
These pages document the country names (official short names in English) in alphabetical order as given in ISO 3166-1 and the corresponding ISO 3166-1-alpha-2 code elements.

[BCP 47 \(Best Current Practice\)](#), [RFC 3066](#)

This document describes a language tag for use in cases where it is desired to indicate the language used in an information object, how to register values for use in this language tag, and a construct for matching such language tags.

[RFC 3282 \(Standards Track\)](#)

This document defines a "Content-language:" header, for use in cases where one desires to indicate the language of something that has RFC 822-like headers, like MIME body parts or Web documents, and an "Accept-Language:" header for use in cases where one wishes to indicate one's preferences with regard to language.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Miscellaneous Documentation](#)

Password Formats

Notes about the password encryption formats generated and understood by Apache.



There are four formats that Apache recognizes for basic-authentication passwords. Note that not all formats work on every platform:

PLAIN TEXT (i.e. *unencrypted*)

Windows, BEOS, & Netware only.

CRYPT

Unix only. Uses the traditional Unix crypt (3) function with a randomly-generated 32-bit salt (only 12 bits used) and the first 8 characters of the password.

SHA1

"{SHA}" + Base64-encoded SHA-1 digest of the password.

MD5

"\$apr1\$" + the result of an Apache-specific algorithm using an iterated (1,000 times) MD5 digest of various combinations of a random 32-bit salt and the password. See the APR source file [apr_md5.c](#) for the details of the algorithm.

Generating values with htpasswd

MD5

```
$ htpasswd -nbm myName myPassword
myName:$apr1$r31.....$HqJZimcKQFAMYayBlzkrA/
```

SHA1

```
$ htpasswd -nbs myName myPassword
myName:{SHA}VBPuJHI7uixaa6LQGwx4s+5GKNE=
```

CRYPT

```
$ htpasswd -nbd myName myPassword
myName:rqXexS6ZhobKA
```

Generating CRYPT and MD5 values with the OpenSSL command-line program

OpenSSL knows the Apache-specific MD5 algorithm.

MD5

```
$ openssl passwd -apr1 myPassword  
$apr1$qHDFfhPC$nITSVHgYbDAK1Y0acGRnY0
```

CRYPT

```
openssl passwd -crypt myPassword  
qQ5vTY03c8dsU
```

Validating CRYPT or MD5 passwords with the OpenSSL command line program

The salt for a CRYPT password is the first two characters (converted to a binary value). To validate myPassword against rqXexS6ZhobKA

CRYPT

```
$ openssl passwd -crypt -salt rq myPassword  
Warning: truncating password to 8 characters  
rqXexS6ZhobKA
```

Note that using myPasswo instead of myPassword will produce the same result because only the first 8 characters of CRYPT passwords are considered.

The salt for an MD5 password is between \$apr1\$ and the following \$ (as a Base64-encoded binary value - max 8 chars). To validate myPassword against \$apr1\$r31.....\$HqJZimcKQFAMYayBlzkrA/

MD5

```
$ openssl passwd -apr1 -salt r31..... myPassword  
$apr1$r31.....$HqJZimcKQFAMYayBlzkrA/
```

Database password fields for mod_dbd

The SHA1 variant is probably the most useful format for DBD authentication. Since the SHA1 and Base64 functions are commonly available, other software can populate a database with encrypted passwords that are usable by Apache basic authentication.

To create Apache SHA1-variant basic-authentication passwords in various languages:

PHP

```
'{SHA}' . base64_encode(sha1($password, TRUE))
```

Java

```
"{SHA}" + new  
sun.misc.BASE64Encoder().encode(java.security.MessageDigest.getInstance
```

ColdFusion

```
"{SHA}" & ToBase64(BinaryDecode(Hash(password, "SHA1"), "Hex"))
```

Ruby

```
require 'digest/sha1'  
require 'base64'  
'{SHA}' + Base64.encode64(Digest::SHA1.digest(password))
```

C or C++

Use the APR function: `apr_sha1_base64`

PostgreSQL (with the contrib/pgcrypto functions installed)

```
'{SHA}' || encode(digest(password, 'sha1'), 'base64')
```



Digest Authentication

Apache recognizes one format for digest-authentication passwords - the MD5 hash of the string `user:realm:password` as a 32-character string of hexadecimal digits. `realm` is the Authorization Realm argument to the [AuthName](#) directive in `httpd.conf`.

Database password fields for `mod_dbd`

Since the MD5 function is commonly available, other software can populate a database with encrypted passwords that are usable by Apache digest authentication.

To create Apache digest-authentication passwords in various languages:

PHP

```
md5($user . ':' . $realm . ':' . $password)
```

Java

```
byte b[] = java.security.MessageDigest.getInstance("MD5").digest(
    (user + ":" + realm + ":" + password ).getBytes());
java.math.BigInteger bi = new java.math.BigInteger(1, b);
String s = bi.toString(16);
while (s.length() < 32)
    s = "0" + s;
// String s is the encrypted password
```

ColdFusion

```
LCASE(Hash( (user & ":" & realm & ":" & password) , "MD5"))
```

Ruby

```
require 'digest/md5'
Digest::MD5.hexdigest(user + ':' + realm + ':' + password)
```

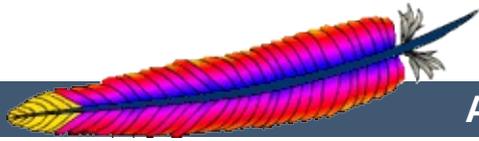
PostgreSQL (with the contrib/pgcrypto functions installed)

```
encode(digest( user || ':' || realm || ':' || password , 'md5'),
```

'hex')

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Terms Used to Describe Modules

This document describes the terms that are used to describe each Apache [module](#).



Description

A brief description of the purpose of the module.



This indicates how tightly bound into the Apache Web server the module is; in other words, you may need to recompile the server in order to gain access to the module and its functionality. Possible values for this attribute are:

MPM

A module with status "MPM" is a [Multi-Processing Module](#). Unlike the other types of modules, Apache must have one and only one MPM in use at any time. This type of module is responsible for basic request handling and dispatching.

Base

A module labeled as having "Base" status is compiled and loaded into the server by default, and is therefore normally available unless you have taken steps to remove the module from your configuration.

Extension

A module with "Extension" status is not normally compiled and loaded into the server. To enable the module and its functionality, you may need to change the server build configuration files and recompile Apache.

Experimental

"Experimental" status indicates that the module is available as part of the Apache kit, but you are on your own if you try to use it. The module is being documented for completeness, and is not necessarily supported.

External

Modules which are not included with the base Apache distribution ("third-party modules") may use the "External" status. We are not responsible for, nor do we support such modules.



This quite simply lists the name of the source file which contains the code for the module. This is also the name used by the `<IfModule>` directive.



This is a string which identifies the module for use in the [LoadModule](#) directive when dynamically loading modules. In particular, it is the name of the external variable of type module in the source file.

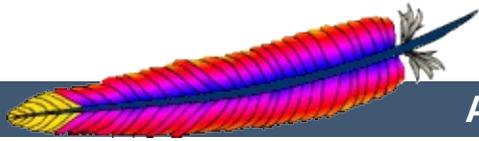


Compatibility

If the module was not part of the original Apache version 2 distribution, the version in which it was introduced should be listed here. In addition, if the module is limited to particular platforms, the details will be listed here.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Terms Used to Describe Directives

This document describes the terms that are used to describe each Apache [configuration directive](#).

See also

[Configuration files](#)



Description

A brief description of the purpose of the directive.



This indicates the format of the directive as it would appear in a configuration file. This syntax is extremely directive-specific, and is described in detail in the directive's definition. Generally, the directive name is followed by a series of one or more space-separated arguments. If an argument contains a space, the argument must be enclosed in double quotes. Optional arguments are enclosed in square brackets. Where an argument can take on more than one possible value, the possible values are separated by vertical bars "|". Literal text is presented in the default font, while argument-types for which substitution is necessary are *emphasized*. Directives which can take a variable number of arguments will end in "..." indicating that the last argument is repeated.

Directives use a great number of different argument types. A few common ones are defined below.

URL

A complete Uniform Resource Locator including a scheme, hostname, and optional pathname as in
`http://www.example.com/path/to/file.html`

URL-path

The part of a *url* which follows the scheme and hostname as in `/path/to/file.html`. The *url-path* represents a web-view of a resource, as opposed to a file-system view.

file-path

The path to a file in the local file-system beginning with the root directory as in `/usr/local/apache/htdocs/path/to/file.html`. Unless otherwise specified, a *file-path* which does not begin with a slash will be treated as relative to the [ServerRoot](#).

directory-path

The path to a directory in the local file-system beginning with the root directory as in `/usr/local/apache/htdocs/path/to/`.

filename

The name of a file with no accompanying path information as in `file.html`.

regex

A Perl-compatible [regular expression](#). The directive definition will specify what the *regex* is matching against.

extension

In general, this is the part of the *filename* which follows the last dot. However, Apache recognizes multiple filename extensions, so if a *filename* contains more than one dot, each dot-separated part of the filename following the first dot is an *extension*. For example, the *filename* `file.html.en` contains two extensions: `.html` and `.en`. For Apache directives, you may specify *extensions* with or without the leading dot. In addition, *extensions* are not case sensitive.

MIME-type

A method of describing the format of a file which consists of a major format type and a minor format type, separated by a slash as in `text/html`.

env-variable

The name of an [environment variable](#) defined in the Apache configuration process. Note this is not necessarily the same as an operating system environment variable. See the [environment variable documentation](#) for more details.



If the directive has a default value (*i.e.*, if you omit it from your configuration entirely, the Apache Web server will behave as though you set it to a particular value), it is described here. If there is no default value, this section should say "*None*". Note that the default listed here is not necessarily the same as the value the directive takes in the default `httpd.conf` distributed with the server.



This indicates where in the server's configuration files the directive is legal. It's a comma-separated list of one or more of the following values:

server config

This means that the directive may be used in the server configuration files (e.g., `httpd.conf`), but **not** within any `<VirtualHost>` or `<Directory>` containers. It is not allowed in `.htaccess` files at all.

virtual host

This context means that the directive may appear inside `<VirtualHost>` containers in the server configuration files.

directory

A directive marked as being valid in this context may be used inside `<Directory>`, `<Location>`, `<Files>`, and `<Proxy>` containers in the server configuration files, subject to the restrictions outlined in [Configuration Sections](#).

.htaccess

If a directive is valid in this context, it means that it can appear inside *per-directory* `.htaccess` files. It may not be processed, though depending upon the [overrides](#) currently active.

The directive is *only* allowed within the designated context; if you try to use it elsewhere, you'll get a configuration error that will either prevent the server from handling requests in that context correctly, or will keep the server from operating at all -- *i.e.*, the server won't even start.

The valid locations for the directive are actually the result of a Boolean OR of all of the listed contexts. In other words, a directive that is marked as being valid in "server config, .htaccess" can be used in the `httpd.conf` file and in `.htaccess` files, but not within any `<Directory>` or `<VirtualHost>` containers.



This directive attribute indicates which configuration override must be active in order for the directive to be processed when it appears in a `.htaccess` file. If the directive's `context` doesn't permit it to appear in `.htaccess` files, then no context will be listed.

Overrides are activated by the [AllowOverride](#) directive, and apply to a particular scope (such as a directory) and all descendants, unless further modified by other [AllowOverride](#) directives at lower levels. The documentation for that directive also lists the possible override names available.



This indicates how tightly bound into the Apache Web server the directive is; in other words, you may need to recompile the server with an enhanced set of modules in order to gain access to the directive and its functionality. Possible values for this attribute are:

Core

If a directive is listed as having "Core" status, that means it is part of the innermost portions of the Apache Web server, and is always available.

MPM

A directive labeled as having "MPM" status is provided by a [Multi-Processing Module](#). This type of directive will be available if and only if you are using one of the MPMs listed on the [Module](#) line of the directive definition.

Base

A directive labeled as having "Base" status is supported by one of the standard Apache modules which is compiled into the server by default, and is therefore normally available unless you've taken steps to remove the module from your configuration.

Extension

A directive with "Extension" status is provided by one of the modules included with the Apache server kit, but the module isn't normally compiled into the server. To enable the directive and its functionality, you will need to change the server build configuration files and re-compile Apache.

Experimental

"Experimental" status indicates that the directive is available as part of the Apache kit, but you're on your own if you try to use it. The directive is being documented for completeness, and is not necessarily supported. The module which provides the directive may or may not be compiled in by default; check the top of the page which describes the directive and its module to see if it remarks on

the availability.



This quite simply lists the name of the source module which defines the directive.

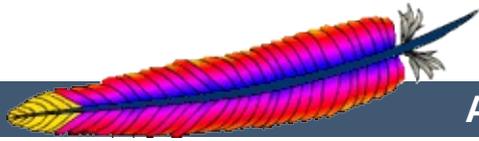


Compatibility

If the directive wasn't part of the original Apache version 2 distribution, the version in which it was introduced should be listed here. In addition, if the directive is available only on certain platforms, it will be noted here.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Apache-Kernfunktionen

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Beschreibung: Ständig verfügbare Kernfunktionen des Apache HTTP Servers

Status: Core



<u>Beschreibung:</u>	Configures optimizations for a Protocol's Listener Sockets
<u>Syntax:</u>	AcceptFilter <i>protocol accept_filter</i>
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Available in Apache 2.1.5 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.

Siehe auch

- [Protocol](#)



Beschreibung:	Ressourcen lassen angehängte Pfadangaben zu
Syntax:	AcceptPathInfo On Off Default
Voreinstellung:	AcceptPathInfo Default
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	FileInfo
Status:	Core
Modul:	core
Kompatibilität:	Verfügbar ab Apache 2.0.30

Die Direktive steuert, ob Anfragen akzeptiert oder abgewiesen werden, bei denen nach der tatsächlichen Datei (oder einer nicht existierenden Datei in einem existierenden Verzeichnis) zusätzliche Pfadangaben folgen. Die angehängte Pfadangabe kann Skripten in der Umgebungsvariable PATH_INFO verfügbar gemacht werden.

Nehmen wir beispielsweise an, dass /test/ auf ein Verzeichnis zeigt, welches lediglich eine Datei here.html enthält. Dann wird bei Anfragen nach /test/here.html/more und /test/nothere.html/more beides Mal /more als PATH_INFO ermittelt.

Die drei möglichen Argumente für die Direktive **AcceptPathInfo** sind:

off

Eine Anfrage wird nur dann akzeptiert, wenn sie exakt auf ein existierendes Verzeichnis (oder eine Datei) abgebildet werden kann. Daher würde eine Anfrage mit einer nach dem tatsächlichen Dateinamen angehängten Pfadangabe, wie /test/here.html/more im obigen Beispiel, den Fehler 404 NOT FOUND (*Anm.d.Ü.:* nicht gefunden) zurückgeben.

On

Eine Anfrage wird akzeptiert, wenn eine vorangestellte Pfadangabe

auf ein existierendes Verzeichnis abgebildet werden kann. Das obige Beispiel `/test/here.html/more` wird akzeptiert, wenn `/test/here.html` auf eine gültige Datei zeigt.

Default

Die Behandlung von Anfragen mit angehängten Pfadangaben wird von dem für die Anfrage verantwortlichen [Handler](#) bestimmt. Der Core-Handler für gewöhnliche Dateien weist `PATH_INFO`-Zugriffe standardmäßig zurück. Handler, die Skripte bedienen, wie z.B. [cgi-script](#) und [isapi-handler](#), sind im Allgemeinen darauf voreingestellt, `PATH_INFO` zu akzeptieren.

Das eigentliche Ziel von `AcceptPathInfo` ist es, Ihnen das Überschreiben der Voreinstellung der Handler bezüglich der Akzeptanz oder Ablehnung von `PATH_INFO` zu erlauben. Eine solche Änderung ist zum Beispiel notwendig, wenn Sie einen [Filter](#) wie [INCLUDES](#) verwenden, um Inhalte abhängig von `PATH_INFO` zu generieren. Der Core-Handler würde die Anfrage normalerweise abweisen. Verwenden Sie die folgende Konfiguration, um dennoch solch ein Skript zu ermöglichen.

```
<Files "mypaths.shtml">
  Options +Includes
  SetOutputFilter INCLUDES
  AcceptPathInfo On
</Files>
```



<u>Beschreibung:</u>	Name der dezentralen Konfigurationsdateien
<u>Syntax:</u>	AccessFileName <i>Dateiname</i> [<i>Dateiname</i>] ...
<u>Voreinstellung:</u>	AccessFileName .htaccess
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core

Aus dieser Namensliste sucht der Server während der Bearbeitung einer Anfrage in jedem Verzeichnis nach der ersten existierenden Datei, sofern im betreffenden Verzeichnis dezentrale Konfigurationsdateien [erlaubt sind](#). Beispiel:

```
AccessFileName .acl
```

Vor der Rücksendung des Dokuments `/usr/local/web/index.html` wird der Server `/.acl`, `/usr/.acl`, `/usr/local/.acl` und `/usr/local/web/.acl` einlesen, solange diese nicht mit

```
<Directory />  
  AllowOverride None  
</Directory>
```

deaktiviert wurden.

Siehe auch

- [AllowOverride](#)
- [Konfigurationsdateien](#)
- [.htaccess-Dateien](#)



<u>Beschreibung:</u>	Standard-Charset-Parameter, der bei Antworten vom Content-Type text/plain oder text/html hinzugefügt wird
<u>Syntax:</u>	AddDefaultCharset On Off <i>Zeichenkodierung</i>
<u>Voreinstellung:</u>	AddDefaultCharset Off
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive gibt einen Standardwert für den Charset-Parameter des Medientyps (den Namen einer Zeichencodierung) an, der einer Antwort genau dann hinzugefügt wird, wenn der Content-Type der Antwort entweder text/plain oder text/html ist. Dies sollte jedes mittels META-Element im Datenteil der Antwort angegebene Charset überschreiben. Das genaue Verhalten hängt jedoch oft von der Client-Konfiguration des Benutzers ab. Die Einstellung AddDefaultCharset Off deaktiviert diese Funktionalität. AddDefaultCharset On aktiviert die Standard-Zeichencodierung iso-8859-1. Jeder andere Wert wird als die zu verwendende *Zeichencodierung* aufgefaßt, die eines der bei [IANA registrierten Charset-Werte](#) zur Verwendung in MIME-Medientypen sein sollte. Zum Beispiel:

```
AddDefaultCharset utf-8
```

AddDefaultCharset sollte nur verwendet werden, wenn von allen Textressourcen, für die es gilt, bekannt ist, dass sie in dieser Zeichencodierung vorliegen, oder wenn es zu unbequem ist, ihre Zeichencodierung individuell zu benennen. Ein solches Beispiel ist das Hinzufügen des Charset-Parameters zu Ressourcen, die generierte

Inhalte enthalten. Ein Beispiel sind CGI-Skript-Altlasten, die aufgrund von in die Ausgabe integrierten Daten, die durch den Benutzer übermittelt wurden, gegen Cross-Site-Scripting-Angriffe verwundbar sind. Eine bessere Lösung wäre jedoch, diese Skripte zu korrigieren (oder zu löschen), da die Angabe einer Standard-Zeichencodierung keine Anwender schützt, die in ihrem Browser die Funktion zur automatischen Erkennung der Zeichenkodierung aktiviert haben.

Siehe auch

- [AddCharset](#)



AddOutputFilterByType Direktive

Beschreibung:	einen Ausgabefilter einem bestimmten MIME-Type zuordnen
Syntax:	AddOutputFilterByType <i>Filter</i> [; <i>Filter</i> ...] <i>MIME-Type</i> [<i>MIME-Type</i>] ...
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	FileInfo
Status:	Core
Modul:	core
Kompatibilität:	Verfügbar ab Apache 2.0.33

Die Direktive aktiviert für eine Anfrage abhängig vom MIME-Type der Antwort einen bestimmten Ausgabe-[Filter](#).

Das folgende Beispiel verwendet den Filter DEFLATE, der von [mod_deflate](#) angeboten wird. Er komprimiert jede Ausgabe, die als text/html oder text/plain gekennzeichnet ist, (gleichgültig, ob statisch oder dynamisch) bevor sie an den Client gesendet wird.

```
AddOutputFilterByType DEFLATE text/html text/plain
```

Wenn Sie den Inhalt von mehr als einem Filter verarbeiten lassen wollen, dann müssen deren Namen durch Semikolons voneinander getrennt werden. Es ist ebenfalls möglich, eine [AddOutputFilterByType](#)-Direktive für jeden von diesen Filtern zu verwenden.

Die folgende Konfiguration sorgt dafür, dass alle Skriptaussagen, die als text/html gekennzeichnet sind, zuerst vom INCLUDES-Filter und dann vom DEFLATE-Filter verarbeitet werden.

```
<Location /cgi-bin/>
  Options Includes
  AddOutputFilterByType INCLUDES;DEFLATE text/html
```

</Location>

Hinweis:

Die Aktivierung von Filtern mittels [AddOutputFilterByType](#) kann in einigen Fällen ganz oder teilweise fehlschlagen. Beispielsweise werden keine Filter angewendet, wenn der MIME-Type nicht bestimmt werden kann und auf die Einstellung der [DefaultType](#)-Anweisung zurückfällt, selbst wenn die [DefaultType](#)-Einstellung die gleiche ist.

Wenn Sie jedoch sicherstellen wollen, dass der Filter angewendet wird, sollten Sie den Content-Type z.B. mit [AddType](#) oder [ForceType](#) der Ressource explizit zuordnen. Das Setzen des Content-Types innerhalb eines (nicht-nph) CGI-Skriptes funktioniert ebenfalls zuverlässig.

Die Typ-gebundenen Ausgabefilter werden niemals auf Proxy-Anfragen angewendet.

Siehe auch

- [AddOutputFilter](#)
- [SetOutputFilter](#)
- [Filter](#)



<u>Beschreibung:</u>	Legt fest, ob kodierte Pfadtrennzeichen in URLs durchgereicht werden dürfen
<u>Syntax:</u>	<code>AllowEncodedSlashes On Off</code>
<u>Voreinstellung:</u>	<code>AllowEncodedSlashes Off</code>
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Verfügbar ab Apache 2.0.46

Die `AllowEncodedSlashes`-Direktive erlaubt die Verwendung von URLs, welche kodierte Pfadtrennzeichen (`%2F` für `/` und auf entsprechenden Systemen zusätzlich `%5C` für `\`) enthalten. Normalerweise werden derartige URLs mit einem 404-Fehler (Nicht gefunden) abgewiesen.

`AllowEncodedSlashes On` ist vor allem in Verbindung mit `PATH_INFO` hilfreich.

Anmerkung

Das Erlauben von Schrägstrichen impliziert *nicht* deren *Dekodierung*. Vorkommen von `%2F` oder `%5C` (*nur* auf entsprechenden Systemen) werden unverändert in der ansonsten dekodierten URL belassen.

Siehe auch

- [AcceptPathInfo](#)



Beschreibung:	Direktiven-Typen, die in <code>.htaccess</code> -Dateien erlaubt sind.
Syntax:	<code>AllowOverride All None Direktiven-Typ [Direktiven-Typ] ...</code>
Voreinstellung:	<code>AllowOverride All</code>
Kontext:	Verzeichnis
Status:	Core
Modul:	core

Wenn der Server eine `.htaccess`-Datei (wie durch [AccessFileName](#) definiert) findet, muss er wissen, welche in der Datei angegebenen Direktiven frühere Konfigurationsanweisungen überschreiben dürfen.

Nur in `<Directory>`-Abschnitten verfügbar

`AllowOverride` ist nur in `<Directory>`-Abschnitten gültig, die ohne reguläre Ausdrücke definiert wurden, nicht in `<Location>`-, `<DirectoryMatch>`- oder `<Files>`-Abschnitten.

Wenn diese Anweisung auf `None` gesetzt wird, dann werden `.htaccess`-Dateien komplett ignoriert. In diesem Fall wird der Server nicht einmal versuchen, die `.htaccess`-Dateien im Dateisystem zu lesen.

Wenn diese Anweisung auf `All` gesetzt wird, dann ist jede Direktive in den `.htaccess`-Dateien erlaubt, die den [Kontext](#) `.htaccess` besitzt.

Der *Direktiven-Typ* kann eine der folgenden Anweisungsgruppen sein.

AuthConfig

Erlaubt die Verwendung von Autorisierungs-Anweisungen ([AuthDBMGroupFile](#), [AuthDBMUserFile](#), [AuthGroupFile](#), [AuthName](#), [AuthType](#), [AuthUserFile](#), [Require](#) usw.).

FileInfo

Erlaubt die Verwendung von Direktiven zur Steuerung der Dokumenttypen ([DefaultType](#), [ErrorDocument](#), [ForceType](#), [LanguagePriority](#), [SetHandler](#), [SetInputFilter](#), [SetOutputFilter](#), und [mod_mime](#)-Direktiven `Add*` und `Remove*` usw.).

Indexes

Erlaubt die Verwendung von Direktiven zur Steuerung von Verzeichnisindizes ([AddDescription](#), [AddIcon](#), [AddIconByEncoding](#), [AddIconByType](#), [DefaultIcon](#), [DirectoryIndex](#), [FancyIndexing](#), [HeaderName](#), [IndexIgnore](#), [IndexOptions](#), [ReadmeName](#) usw.).

Limit

Erlaubt die Verwendung von Direktiven zur Steuerung des Zugriffs von Hosts ([Allow](#), [Deny](#) und [Order](#)).

Options[=*Option*,...]

Erlaubt die Verwendung von Direktiven zur Steuerung spezieller Verzeichniseigenschaften ([Options](#) und [XBitHack](#)). Sie können mit einem Gleichheitszeichen gefolgt von einer kommaseparierten Liste (ohne Leerzeichen) angeben, welche Optionen mit der [Options](#)-Direktive gesetzt werden dürfen.

Beispiel:

```
AllowOverride AuthConfig Indexes
```

Im obigen Beispiel erzeugen alle Direktiven einen internal server error (*Anm.d.Ü.:* Server-interner Fehler), die weder der Gruppe `AuthConfig` noch der Gruppe `Indexes` angehören.

Siehe auch

- [AccessFileName](#)
- [Konfigurationsdateien](#)

- [.htaccess-Dateien](#)



<u>Beschreibung:</u>	Autorisierungsbereich zur Verwendung in der HTTP-Authentisierung
<u>Syntax:</u>	AuthName <i>auth-Bereich</i>
<u>Kontext:</u>	Verzeichnis, .htaccess
<u>AllowOverride:</u>	AuthConfig
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive legt den Namen des Autorisierungsbereiches (*Anm.d.Ü.:* Der Autorisierungsbereich wird auch Realm genannt.) für ein Verzeichnis fest. Dieser Realm wird dem Client mitgeteilt, damit der Anwender weiß, welchen Benutzernamen und welches Passwort er zu übermitteln hat.

AuthName akzeptiert ein Argument. Falls der Name des Realm Leerzeichen enthält, muss er in Anführungszeichen eingeschlossen werden. Um zu funktionieren, muss die Anweisung von den Direktiven **AuthType** und **Require** sowie von Direktiven wie **AuthUserFile** und **AuthGroupFile** begleitet werden.

Beispiel:

```
AuthName "Top Secret"
```

Die AuthName übergebene Zeichenkette ist das, was in dem von den meisten Browsern angebotenen Passwort-Dialog angezeigt wird.

Siehe auch

- [Authentisierung, Autorisierung und Zugriffskontrolle](#)



Beschreibung:	Art der Authentisierung
Syntax:	AuthType Basic Digest
Kontext:	Verzeichnis, .htaccess
AllowOverride:	AuthConfig
Status:	Core
Modul:	core

Die Direktive wählt die Art der Benutzer-Authentisierung für ein Verzeichnis aus. Derzeit sind lediglich Basic und Digest implementiert. Um zu funktionieren, muss die Anweisung von den Direktiven [AuthName](#) und [Require](#) sowie von Direktiven wie [AuthUserFile](#) und [AuthGroupFile](#) begleitet werden.

Siehe auch

- [Authentisierung, Autorisierung und Zugriffskontrolle](#)



<u>Beschreibung:</u>	Technik zur Bestimmung des Interpreters für CGI-Skripte
<u>Syntax:</u>	CGIMapExtension <i>CGI-Pfad</i> <i>.Endung</i>
<u>Kontext:</u>	Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	ausschließlich NetWare

Die Direktive wird zur Steuerung verwendet, wie Apache den Interpreter ermittelt, der zur Ausführung von CGI-Skripten verwendet wird. Beispielsweise bestimmt die Angabe von CGIMapExtension `sys:\foo.nlm .foo`, dass alle CGI-Skripte mit der Endung `.foo` an den FOO-Interpreter übergeben werden.



<u>Beschreibung:</u>	Aktiviert die Generierung von Content -MD5 HTTP-Response-Headern
<u>Syntax:</u>	ContentDigest On Off
<u>Voreinstellung:</u>	ContentDigest Off
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	Options
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive aktiviert die Generierung von Content -MD5-Headern, wie sie in RFC1864 bzw. RFC2616 definiert sind.

MD5 ist ein Algorithmus zur Berechnung eines "Datenextrakts" (zuweilen "Fingerabdruck" genannt) (*Anm.d.Ü.:* Der "Datenextrakt" wird im Englischen als "message digest" oder "fingerprint" bezeichnet.) aus beliebig langen Daten. Es gilt als zuverlässig, dass Veränderungen an den Daten sich in Veränderungen des Extrakts widerspiegeln.

Der Content -MD5-Header bietet eine End-to-End-Integritätsprüfung (MIC) (*Anm.d.Ü.:* MIC steht für "message integrity check".) des Daten-Inhalts. Ein Proxy oder Client kann diesen Header prüfen, um zufällige Veränderungen des Entity-Inhalts bei der Übertragung festzustellen.

Beispielheader:

```
Content-MD5: AuLb7Dp1rqtRtxz2m9kRpA==
```

Beachten Sie bitte, dass dies Performanceprobleme auf Ihrem System verursachen kann, da der Extrakt bei jeder Anfrage berechnet wird (der Wert wird nicht zwischengespeichert).

Content -MD5 wird nur für Dokumente gesendet, die von **core** bedient werden, nicht jedoch bei Modulen. SSI-Dokumente, CGI-Skript-

Ausgaben und Byte-Range-Antworten besitzen diesen Header beispielsweise nicht.



DefaultType-Direktive

Beschreibung:	MIME-Content-Type, der gesendet wird, wenn der Server den Typ nicht auf andere Weise ermitteln kann.
Syntax:	DefaultType <i>MIME-Type</i>
Voreinstellung:	DefaultType text/plain
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	FileInfo
Status:	Core
Modul:	core

Es kann vorkommen, dass der Server ein Dokument ausliefern muss, dessen Typ er nicht mit Hilfe seiner MIME-Type-Zuordnungen bestimmen kann.

Der Server muss den Client über den Content-Type des Dokumentes informieren. Daher verwendet er im Falle eines unbekanntens Typs die DefaultType-Einstellung. Zum Beispiel:

```
DefaultType image/gif
```

wäre angemessen für ein Verzeichnis, das viele GIF-Bilder enthält, deren Dateinamen nicht Endung `.gif` besitzen.

Beachten Sie bitte, dass die Direktive anders als [ForceType](#) lediglich den Standard-MIME-Type bestimmt. Alle anderen MIME-Type-Definitionen, einschließlich Dateierweiterungen, die den Medien-Typ anzeigen können, überschreiben diese Voreinstellung.



Beschreibung:	Umschließt eine Gruppe von Direktiven, die nur auf das genannte Verzeichnis des Dateisystems und Unterverzeichnisse angewendet werden
Syntax:	<code><Directory Verzeichnispfad> ... </Directory></code>
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

`<Directory>` und `</Directory>` werden dazu verwendet, eine Gruppe von Direktiven zusammenzufassen, die nur für das genannte Verzeichnis und dessen Unterverzeichnisse gelten. Jede Direktive, die im Verzeichnis-Kontext erlaubt ist, kann verwendet werden. *Verzeichnispfad* ist entweder der vollständige Pfad zu einem Verzeichnis oder eine Zeichenkette mit Platzhaltern wie sie von der Unix-Shell zum Abgleich verwendet werden. In einer Zeichenkette mit Platzhaltern (*Anm.d.Ü.: sogenannte wild-cards*) entspricht `?` einem einzelnen Zeichen und `*` einer Zeichenkette beliebiger Länge. Sie können auch auch `[]`-Zeichenbereiche verwenden. Keiner der Platzhalter entspricht dem Zeichen `/`. Daher passt `<Directory /*/public_html>` nicht auf `/home/user/public_html`, `<Directory /home/*/public_html>` jedoch tut es. Beispiel:

```
<Directory /usr/local/httpd/htdocs>
  Options Indexes FollowSymLinks
</Directory>
```

Seien Sie vorsichtig mit den *Verzeichnispfad*-Argumenten. Sie müssen buchstäblich mit dem Dateisystempfad übereinstimmen, den der Apache für den Zugriff auf die Dateien verwendet. Direktiven, die für ein bestimmtes Verzeichnis gelten, gelten nicht für Dateien in dem Verzeichnis, auf die über einen anderen Pfad zugegriffen wird, wie z.B.

über verschiedene symbolische Links.

Erweiterte reguläre Ausdrücke können ebenfalls verwendet werden, indem das Zeichen ~ hinzugefügt wird. Beispielsweise würde

```
<Directory ~ "^/www/.*/[0-9]{3}">
```

auf Verzeichnisse in /www/ passen, die aus drei Zahlen bestehen.

Wenn mehrere `<Directory>`-Abschnitte (ohne reguläre Ausdrücke) auf ein Verzeichnis (oder ein ihm übergeordnetes Verzeichnis) passen, welches ein Dokument enthält, dann werden die Direktiven der Reihe nach, angefangen beim kürzesten passenden Muster, vermischt mit den Direktiven aus den `.htaccess`-Dateien, angewendet. Beispiel:

```
<Directory />
  AllowOverride None
</Directory>

<Directory /home/>
  AllowOverride FileInfo
</Directory>
```

Beim Zugriff auf das Dokument `/home/web/dir/doc.html` sind die einzelnen Schritte:

- Wende die Direktive `AllowOverride None` an (deaktiviere `.htaccess`-Dateien).
- Wende die Direktive `AllowOverride FileInfo` (auf das Verzeichnis `/home`) an.
- Wende jede `FileInfo`-Direktive aus `/home/.htaccess`, `/home/web/.htaccess` und `/home/web/dir/.htaccess` der Reihe nach an.

Reguläre Ausdrücke werden solange nicht berücksichtigt, bis alle normalen Abschnitte angewendet wurden. Anschließend werden alle

regulären Ausdrücke in der Reihenfolge geprüft, in der sie in der Konfigurationsdatei auftauchen. Beispielsweise wird bei

```
<Directory ~ abc$>
  # ... hier die Direktiven ...
</Directory>
```

der Abschnitt mit dem regulären Ausdruck nicht berücksichtigt, bis alle normalen `<Directory>`-Abschnitte und `.htaccess`-Dateien angewendet wurden. Dann erst wird der reguläre Ausdruck mit `/home/abc/public_html/abc` abgeglichen und der entsprechende `<Directory>`-Abschnitt angewendet.

Beachten Sie bitte, dass der vom Apache voreingestellte Zugriff für `<Directory /> Allow from All` ist. Das bedeutet, dass der Apache jede Datei ausliefert, die durch eine URL abgebildet wird. Es wird empfohlen, dass Sie dies durch einen Block wie

```
<Directory />
  Order Deny,Allow
  Deny from All
</Directory>
```

ändern und anschließend für Verzeichnisse überschreiben, die Sie verfügbar machen *wollen*. Für weitere Einzelheiten lesen Sie bitte die Seite zu den [Sicherheitshinweisen](#).

Die Verzeichnisabschnitte erscheinen in der Datei `httpd.conf`. `<Directory>`-Direktiven dürfen nicht ineinander verschachtelt werden oder innerhalb von `<Limit>`- oder `<LimitExcept>`-Abschnitten auftauchen.

Siehe auch

- [Wie die Abschnitte `<Directory>`, `<Location>` und `<Files>` arbeiten](#) für eine Erläuterung, wie diese verschiedenen Abschnitte miteinander

kombiniert werden, wenn eine Anfrage empfangen wird



Beschreibung:	Umschließt eine Gruppe von Direktiven, die auf Verzeichnisse des Dateisystems und ihre Unterverzeichnisse abgebildet werden, welche auf einen regulären Ausdruck passen
Syntax:	<code><DirectoryMatch regex> ...</code> <code></DirectoryMatch></code>
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

`<DirectoryMatch>` und `</DirectoryMatch>` werden dazu verwendet, eine Gruppe von Direktiven zusammenzufassen, die nur für das genannte Verzeichnis und dessen Unterverzeichnisse gelten, genauso wie bei `<Directory>`. Als Argument dient jedoch ein regulärer Ausdruck. Beispielsweise würde

```
<DirectoryMatch "^/www/.*/[0-9]{3}">
```

auf Verzeichnisse in `/www/` passen, die aus drei Zeichen bestehen.

Siehe auch

- [<Directory>](#) für eine Beschreibung, wie reguläre Ausdrücke mit normalen `<Directory>`-Anweisungen vermischt werden.
- [Wie die Abschnitte <Directory>, <Location> und <Files> arbeiten](#) für eine Erläuterung, wie diese verschiedenen Abschnitte miteinander kombiniert werden, wenn eine Anfrage empfangen wird



<u>Beschreibung:</u>	Verzeichnis, welches den Haupt-Dokumentenbaum bildet, der im Web sichtbar ist.
<u>Syntax:</u>	DocumentRoot <i>Verzeichnis</i>
<u>Voreinstellung:</u>	DocumentRoot /usr/local/apache/htdocs
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive setzt das Verzeichnis, von dem aus [httpd](#) Dateien ausliefert. Sofern nicht eine Direktive wie [Alias](#) greift, hängt der Server Pfade aus der angeforderten URL an das Wurzelverzeichnis an, um den Pfad zum Dokument zu bilden. Beispiel:

```
DocumentRoot /usr/web
```

Damit bezieht sich ein Zugriff auf `http://www.my.host.com/index.html` auf `/usr/web/index.html`. Wenn das *Verzeichnis* nicht absolut angegeben ist, wird es relativ zu [ServerRoot](#) betrachtet.

[DocumentRoot](#) sollte ohne einen Schrägstrich am Ende angegeben werden.

Siehe auch

- [URLs auf das Dateisystem abbilden](#)



<u>Beschreibung:</u>	Verwende Memory-Mapping, um Dateien während der Auslieferung zu lesen
<u>Syntax:</u>	EnableMMAP On Off
<u>Voreinstellung:</u>	EnableMMAP On
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive steuert, ob [httpd](#) Memory-Mapping verwenden darf, wenn er während der Auslieferung den Inhalt einer Datei lesen muss. Wenn die Bearbeitung einer Anfrage es erfordert, auf die Daten in einer Datei zuzugreifen -- zum Beispiel bei der Auslieferung einer mittels [mod_include](#) serverseitig analysierten Datei --, dann verwendet der Apache standardmäßig Memory-Mapping für diese Datei, sofern das Betriebssystem es unterstützt.

Memory-Mapping bedeutet zuweilen eine Performanceverbesserung. In einigen Umgebungen ist es jedoch besser, Memory-Mapping zu deaktivieren, um Problemen während des Betriebs vorzubeugen:

- Bei einigen Multiprozessorsystemen kann Memory-Mapping die Performance von [httpd](#) reduzieren.
- Bei einem per NFS eingebundenen [DocumentRoot](#) kann [httpd](#) mit einem Speicherzugriffsfehler (*Anm.d.Ü.:* ein so genannter "segmentation fault") abstürzen, wenn eine Datei gelöscht oder gekürzt wird, während [httpd](#) sie im Speicher abbildet.

Bei Serverkonfigurationen, die für dieses Problem anfällig sind, sollten Sie das Memory-Mapping für auszuliefernde Dateien deaktivieren, indem Sie schreiben:

```
EnableMMAP Off
```

Bei per NFS eingebundenen Dateien kann diese Funktion explizit für die störenden Dateien deaktiviert werden, indem Sie angeben:

```
<Directory "/pfad-zu-den-nfs-dateien">  
  EnableMMAP Off  
</Directory>
```



<u>Beschreibung:</u>	Verwende die sendfile-Unterstützung des Kernels, um Dateien an den Client auszuliefern
<u>Syntax:</u>	EnableSendfile On Off
<u>Voreinstellung:</u>	EnableSendfile On
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Verfügbar ab Apache Version 2.0.44

Die Direktive steuert, ob [httpd](#) die sendfile-Unterstützung des Kernels verwenden kann, um Dateiinhalte an den Client zu übermitteln. Wenn die Bearbeitung einer Anfrage keinen Zugriff auf die Daten in der Datei erfordert -- zum Beispiel bei der Auslieferung einer statischen Datei -- und das Betriebssystem es unterstützt, verwendet der Apache standardmäßig sendfile, um den Dateiinhalt zu übertragen, ohne die Datei jemals zu lesen.

Der sendfile-Mechanismus vermeidet getrennte Lese- und Sendeoperationen sowie Puffer-Zuweisungen. Bei einigen Plattformen bzw. Dateisystemen deaktivieren Sie diese Funktion jedoch besser, um Probleme während des Betriebs zu vermeiden:

- Einige Plattformen besitzen u.U. eine fehlerhafte sendfile-Unterstützung, die das Erstellungssystem nicht erkennt, insbesondere wenn die Binärdateien auf einem anderen Rechner erstellt und auf eine solche Maschine mit fehlerhafter sendfile-Unterstützung übertragen wurden.
- Bei einem über das Netzwerk eingebundenen [DocumentRoot](#) (z.B. NFS oder SMB) ist der Kernel möglicherweise nicht in der Lage, die Netzwerkdatei über seinen eigenen Cache zu bedienen.

- Unter Linux löst die Verwendung von `sendfile` in Verbindung mit bestimmten Netzwerkkarten und IPv6 TCP-Checksummenfehler aus.

Bei Serverkonfigurationen, die für dieses Problem anfällig sind, sollten die diese Funktion deaktivieren, indem Sie schreiben:

```
EnableSendfile Off
```

Bei per NFS oder SMB eingebundenen Dateien kann diese Funktion explizit für die störenden Dateien deaktiviert werden, indem Sie angeben:

```
<Directory "/pfad-zu-den-nfs-dateien">  
  EnableSendfile Off  
</Directory>
```

Beachten Sie bitte, dass die verzeichnisbasierte und `.htaccess`-Konfiguration von `EnableSendfile` nicht vom `mod_disk_cache`-Modul unterstützt wird. Nur die globale Konfiguration von `EnableSendfile` wird vom Modul beachtet.



<u>Beschreibung:</u>	Das, was der Server im Fehlerfall an den Client zurückgibt
<u>Syntax:</u>	ErrorDocument <i>Fehlercode Dokument</i>
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Die Syntax der Anführungszeichen bei Textnachrichten hat sich im Apache 2.0 geändert

Im Falle eines Problems oder Fehlers kann der Apache konfiguriert werden, eine der vier Aktionen auszuführen:

1. Ausgabe einer einfachen, hartkodierten Fehlermeldung
2. Ausgabe einer angepassten Meldung
3. Umleitung zu einem lokalen *URL-Pfad* der das Problem bzw. den Fehler behandelt
4. Umleitung zu einer externen *URL*, die das Problem bzw. den Fehler behandelt

Die erste Option ist Voreinstellung, während die Optionen 2 bis 4 über die Direktive **ErrorDocument** eingestellt werden, welcher der HTTP-Statuscode und eine URL oder Nachricht folgen. Abhängig vom Problem bzw. Fehler bietet der Apache manchmal zusätzliche Informationen an.

URLs können bei lokalen Webpfaden mit einem Schrägstrich (/) beginnen (relativ zum **DocumentRoot**-Verzeichnis) oder eine vollständige URL bilden, die der Client auflösen kann. Alternativ kann eine Nachricht für die Anzeige im Browser angeboten werden. Beispiel:

```
ErrorDocument 500 http://foo.example.com/cgi-bin/tester
```

```
ErrorDocument 404 /cgi-bin/falsche_urls.pl
ErrorDocument 401 /info_zur_anmeldung.html
ErrorDocument 403 "Der Zugriff ist nicht erlaubt."
```

Außerdem kann der spezielle Wert `default` angegeben werden, um die schlichte, hartkodierte Nachricht des Apache zu verwenden. Es wird normalerweise nicht benötigt, doch `default` stellt die einfach, im Apache hartkodierte Meldung in Konfigurationen wieder her, die ansonsten von einem existierenden (*Anm.d.Ü.:* zuvor konfigurierten) **ErrorDocument** erben würden.

```
ErrorDocument 404 /cgi-bin/bad_urls.pl

<Directory /web/docs>
    ErrorDocument 404 default
</Directory>
```

Wenn Sie eine **ErrorDocument**-Anweisung angeben, die auf eine entfernte URL weist (d.h. irgendetwas mit der Methode `http` davor), beachten Sie bitte, dass der Apache eine Umleitung zum Client sendet, um diesem mitzuteilen, wo das Dokument zu finden ist, auch wenn das Dokument letztlich wieder zum gleichen Server führt. Das hat mehrere Auswirkungen. Die wichtigste ist, dass der Client nicht den Original-Statuscode erhält sondern statt dessen einen Umleitungs-Statuscode. Dies wiederum kann Web-Robots und andere Clients verwirren, die den Statuscode dazu verwenden, herauszufinden ob eine URL gültig ist. Wenn Sie eine entfernte URL in einer Anweisung `ErrorDocument 401` verwenden, wird der Client darüber hinaus nicht wissen, dass er den Benutzer zur Eingabe eines Passwortes auffordern muss, da er den Statuscode 401 nicht erhält. **Deshalb müssen Sie sich auf ein lokales Dokument beziehen, wenn Sie eine Anweisung `ErrorDocument 401` verwenden.**

Der Microsoft Internet Explorer (MSIE) ignoriert standardmäßig serverseitig generierte Fehlermeldungen, wenn sie "zu kurz" sind und ersetzt sie durch eigene "freundliche" Fehlermeldungen. Die Größe

variiert abhängig von der Art des Fehlers, im Allgemeinen zeigt der MSIE jedoch den serverseitig generierten Fehler, anstatt ihn zu verstecken, wenn Ihr Fehlerdokument größer als 512 Bytes ist. Weitere Informationen sind im Artikel [Q294807](#) in der Microsoft Knowledgebase verfügbar.

Obwohl die meisten Fehlermeldungen überschrieben werden können, werden unter bestimmten Umständen die internen Meldungen ungeachtet der Einstellung der `ErrorDocument`-Direktive verwendet. Insbesondere bei einer fehlerhaften Anfrage werden der normale Bearbeitungsprozess sofort beendet und die interne Meldung zurückgegeben. Das ist notwendig, um Sicherheitsprobleme zu vermeiden, die auf Grund fehlerhafter Anfragen entstehen.

In Versionen vor 2.0 wurden Meldungen durch ein einzelnes vorangestelltes Anführungszeichen (") erkannt.

Siehe auch

- [Dokumentation zu individuellen Fehlermeldungen](#)



Beschreibung:	Ablageort, an dem der Server Fehler protokolliert
Syntax:	<code>ErrorLog <i>Dateiname</i> syslog[:<i>facility</i>]</code>
Voreinstellung:	<code>ErrorLog logs/error_log</code> (Unix) <code>ErrorLog logs/error.log</code> (Windows and OS/2)
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

Die Direktive `ErrorLog` bestimmt den Namen der Datei, in welcher der Server alle auftretenden Fehler protokolliert. Wenn *Dateiname* nicht absolut ist, wird er relativ zu `ServerRoot` betrachtet.

Beispiel

```
ErrorLog /var/log/httpd/error_log
```

Wenn der *Dateiname* mit einem senkrechten Strich (`|`, engl.: Pipe) beginnt, wird angenommen, dass es sich um einen Befehl handelt, der ausgeführt wird, um das Fehlerprotokolls zu verarbeiten.

Beispiel

```
ErrorLog "|/usr/local/bin/httpd_errors"
```

Die Verwendung von `syslog` anstelle eines Dateinamens aktiviert die Protokollierung mittels `syslogd(8)`, sofern das System es unterstützt. Als Voreinstellung wird der `syslog`-Typ (`syslog facility`) `local7` verwendet, Sie können dies jedoch auch überschreiben, indem Sie die Syntax `syslog:facility` verwenden, wobei *facility* einer der Namen sein kann, die üblicherweise in `syslog(1)` dokumentiert sind.

Beispiel

```
ErrorLog syslog:user
```

SICHERHEITSHINWEIS: Lesen Sie das Dokument [Sicherheitshinweise](#) zu Einzelheiten darüber, warum Ihre Sicherheit gefährdet sein kann, wenn das Verzeichnis, in dem die Log-Dateien gespeichert werden, für jemand anderen, als den Benutzer, der den Server gestartet hat, beschreibbar ist.

Anmerkung

Bei der Eingabe eines Dateipfads auf nicht-Unix-Plattformen sollte darauf geachtet werden, nur (Vorwärts-)Schrägstriche zu verwenden, auch wenn die Plattform rückwärts gerichtete Schrägstriche (Backslashes) erlaubt. Im Allgemeinen ist es eine gute Idee, innerhalb der Konfigurationsdateien immer Vorwärts-Schrägstriche zu verwenden.

Siehe auch

- [LogLevel](#)
- [Apache-Log-Dateien](#)



Beschreibung:	Dateiattribute, die zur Erstellung des HTTP-Response-Headers ETag verwendet werden
Syntax:	<code>FileETag Komponente ...</code>
Voreinstellung:	<code>FileETag INode MTime Size</code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	FileInfo
Status:	Core
Modul:	core

Wenn dem Dokument eine Datei zugrundeliegt, bestimmt die Direktive `FileETag` die Dateiattribute, die zur Erstellung des HTTP-Response-Headers ETag (Entity-Tag) verwendet werden. (Der Wert von ETag wird bei der Cache-Verwaltung zur Einsparung von Netzwerk-Bandbreite benutzt.) Im Apache 1.3.22 und früher wurde der ETag-Wert `stets` aus der I-Node, der Größe und dem Datum der letzten Änderung (`mtime`) der Datei gebildet. Die Direktive `FileETag` erlaubt es Ihnen, zu bestimmen, welche dieser Eigenschaften -- falls überhaupt -- verwendet werden sollen. Die gültigen Schlüsselworte lauten:

INode

Die I-Node-Nummer wird in die Berechnung mit einbezogen

MTime

Datum und Uhrzeit der letzten Änderung werden mit einbezogen

Size

Die Anzahl der Bytes in der Datei wird mit einbezogen

All

Alle verfügbaren Angaben werden verwendet. Die ist gleichbedeutend mit:

```
FileETag INode MTime Size
```

None

Es wird keine ETag-Angabe in die Antwort eingefügt, wenn dem Dokument eine Datei zugrundeliegt.

Den Schlüsselwörtern `INode`, `MTime` und `Size` kann entweder ein `+` oder ein `-` vorangestellt werden, was die Änderung einer Vorgabe erlaubt, die von einem größeren Umfeld geerbt wurde. Jedes Schlüsselwort ohne ein solches Prefix hebt die ererbte Einstellung sofort und vollständig auf.

Wenn die Konfiguration für ein Verzeichnis `FileETag INode MTime Size` enthält und die eines Unterverzeichnisses `FileETag -INode`, dann ist die Einstellung für das Unterverzeichnis (die an jedes Unter-Unterverzeichnis weitervererbt wird, welches dies nicht überschreibt) äquivalent mit `FileETag MTime Size`.



Beschreibung:	Enthält Direktiven, die sich nur auf passende Dateinamen beziehen
Syntax:	<code><Files <i>Dateiname</i>> ... </Files></code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	All
Status:	Core
Modul:	core

Die Direktive `<Files>` begrenzt die Reichweite der enthaltenen Anweisungen auf Dateinamen. Sie ist vergleichbar mit den Direktiven `<Directory>` und `<Location>`. Sie muss eine passende `</Files>`-Anweisung besitzen. Die innerhalb dieses Abschnittes angegebenen Direktiven werden auf jedes Objekt mit einem Basisnamen (letzte Komponente des Dateinamens) angewendet, der auf die angegebenen Dateinamen passt. `<Files>`-Container werden, nachdem die `<Directory>`-Container und .htaccess-Dateien gelesen sind, jedoch vor den `<Location>`-Containern, in der Reihenfolge ihres Auftretens ausgeführt. Beachten Sie, dass `<Files>`-Anweisungen innerhalb von `<Directory>`-Containern auftreten können, um den Teil des Dateisystems einzuschränken, den sie betreffen.

Das Argument *Dateiname* kann einen Dateinamen oder eine Zeichenkette mit Platzhaltern enthalten, wobei ? auf ein einzelnes Zeichen passt und * auf eine beliebige Folge von Zeichen. Erweiterte reguläre Ausdrücke können ebenfalls verwendet werden, indem das Zeichen ~ hinzugefügt wird. Beispielsweise würde

```
<Files ~ "\.(gif|jpe?g|png)$">
```

auf die gebräuchlichsten Grafikformate im Internet passen. `<FilesMatch>` wird jedoch bevorzugt.

Beachten Sie bitte, dass die `<Files>`-Container anders als `<Directory>`- und `<Location>`-Container innerhalb von `.htaccess`-Dateien verwendet werden können. Dies erlaubt den Anwendern auf Dateiebene die Kontrolle über ihre eigenen Dateien.

Siehe auch

- [Wie die Abschnitte `<Directory>`, `<Location>` und `<Files>` arbeiten](#) für eine Erläuterung, wie diese verschiedenen Abschnitte miteinander kombiniert werden, wenn eine Anfrage empfangen wird



<u>Beschreibung:</u>	Enthält Direktiven, die für Dateinamen gelten, die auf einen regulären Ausdruck passen
<u>Syntax:</u>	<code><FilesMatch regex> ... </FilesMatch></code>
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	All
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive `<FilesMatch>` begrenzt wie die Direktive `<Files>` die enthaltenen Anweisungen auf Dateinamen. Sie akzeptiert jedoch reguläre Ausdrücke. Beispielsweise würde

```
<FilesMatch "\.(gif|jpe?g|png)$">
```

auf die gebräuchlichsten Grafikformate im Internet passen.

Siehe auch

- [Wie die Abschnitte <Directory>, <Location> und <Files> arbeiten](#) für eine Erläuterung, wie diese verschiedenen Abschnitte miteinander kombiniert werden, wenn eine Anfrage empfangen wird



Beschreibung:	Erzwingt die Auslieferung aller passenden Dateien mit dem angegebenen MIME-Content-Type
Syntax:	ForceType <i>MIME-Type</i> None
Kontext:	Verzeichnis, .htaccess
AllowOverride:	FileInfo
Status:	Core
Modul:	core
Kompatibilität:	Wurde im Apache 2.0 in den Core verschoben

Wenn sie innerhalb einer .htaccess-Datei, eines `<Directory>`-, `<Location>`- `<Files>`-Containers angegeben wird, erzwingt die Direktive die Auslieferung aller entsprechenden Dateien mit dem Content-Type, der durch *MIME-Type* definiert wurde. Wenn Sie zum Beispiel ein Verzeichnis voller GIF-Dateien haben, die Sie nicht alle durch `.gif` kennzeichnen wollen, können Sie angeben:

```
ForceType image/gif
```

Beachten Sie bitte, dass die Direktive anders als `DefaultType` alle MIME-Type-Zuordnungen überschreibt, einschließlich Dateiendungen, die einen Medientyp bezeichnen könnten.

Sie können jede `ForceType`-Angabe durch die Verwendung des Wertes `None` überschreiben:

```
# erzwinge image/gif für alle Dateien:
<Location /images>
    ForceType image/gif
</Location>

# hier jedoch normale MIME-Type-Zuordnungen:
<Location /images/mixed>
    ForceType None
</Location>
```



Beschreibung:	Directory to write gmon.out profiling data to.
Syntax:	GprofDir /tmp/gprof/ /tmp/gprof/%
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



Beschreibung:	Aktiviert DNS-Lookups auf Client-IP-Adressen
Syntax:	HostnameLookups On Off Double
Voreinstellung:	HostnameLookups Off
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis
Status:	Core
Modul:	core

Diese Direktive aktiviert die DNS-Abfrage (*Anm.d.Ü.:* ein sogenannter DNS-Lookup), so dass Hostnamen protokolliert (und in REMOTE_HOST an CGIs/SSIs übergeben) werden können. Der Wert Double bezieht sich auf ein Double-Reverse-DNS-Lookup. D.h. nachdem ein Reverse-Lookup durchgeführt wurde, wird dann auf dem Ergebnis ein Forward-Lookup ausgeführt. Wenigstens eine der IP-Adressen aus dem Forward-Lookup muss der Originaladresse entsprechen. (In der "tcpwrappers"-Terminologie wird dies PARANOID genannt.)

Unabhängig von der Einstellung wird ein Double-Reverse-Lookup durchgeführt, wenn [mod_auth_host](#) zur Zugriffskontrolle per Hostnamen eingesetzt wird. Dies ist aus Sicherheitsgründen notwendig. Beachten Sie, dass das Ergebnis dieses Double-Reverse-Lookups nicht generell verfügbar ist, solange Sie nicht HostnameLookups Double setzen. Wenn beispielsweise nur HostnameLookups On angegeben ist und eine Anfrage für ein Objekt erfolgt, welches durch Hostnamen-Beschränkungen geschützt ist, dann wird CGIs nur das Ergebnis des Singel-Reverse-Lookups in REMOTE_HOST übergeben, egal ob das Doble-Reverse-Lookup fehlschlug oder nicht.

Die Voreinstellung ist Off, um Netzwerktraffic bei den Angeboten einzusparen, die nicht tatsächlich Reverse-Lookups benötigen. Es ist auch für die Endanwender besser, da sie nicht die zusätzliche Wartezeit ertragen müssen, die ein Lookup mit sich bringt. Hoch frequentierte Angebote sollten diese Direktive auf Off fassen. Das Hilfsprogramm

[logresolve](#), das standardmäßig in das Unterverzeichnis bin Ihres Installationsverzeichnis kompiliert wird, kann dazu verwendet werden, um offline Hostnamen von protokollierten IP-Adressen nachzuschlagen.



<u>Beschreibung:</u>	Modify restrictions on HTTP Request Messages
<u>Syntax:</u>	HttpProtocolOptions [Strict Unsafe] [RegisteredMethods LenientMethods] [Allow0.9 Require1.0]
<u>Voreinstellung:</u>	HttpProtocolOptions Strict LenientMethods Allow0.9
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	2.2.32 or 2.4.24 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



Beschreibung:	Schließt Direktiven ein, die nur ausgeführt werden, wenn eine Testbedingung beim Start wahr ist
Syntax:	<code><IfDefine [!]Parametername> ... </IfDefine></code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	All
Status:	Core
Modul:	core

Der Container `<IfDefine Test>...</IfDefine>` wird dazu verwendet, Direktiven als bedingt zu kennzeichnen. Die Direktiven innerhalb eines `<IfDefine>`-Abschnittes werden nur ausgeführt, wenn *Test* wahr ist. Ist *Test* falsch, wird alles zwischen der Start- und Endemarkierung ignoriert.

In der `<IfDefine>`-Anweisung kann *Test* eine von zwei Formen annehmen:

- *Parametername*
- *!Parametername*

Im ersten Fall werden die Direktiven zwischen der Start- und Endemarkierung nur ausgeführt, wenn der Parameter namens *Parametername* definiert ist. Die zweite Form kehrt den Test um und führt die Direktiven nur dann aus, wenn *Parametername* **nicht** definiert ist.

Das Argument *Parametername* ist ein sogenanntes "Define", das beim Start des Servers in der `httpd`-Befehlszeile durch `-DParameter` angegeben wird.

`<IfDefine>`-Container können ineinander verschachtelt werden, um einfache Multi-Parameter-Tests zu implementieren. Beispiel:

```
httpd -DReverseProxy ...
```

```
# httpd.conf
```

```
<IfDefine ReverseProxy>
```

```
    LoadModule rewrite_module modules/mod_rewrite.so
```

```
    LoadModule proxy_module modules/libproxy.so
```

```
</IfDefine>
```



Beschreibung:	Schließt Direktiven ein, die abhängig vom Vorhandensein oder Fehlen eines speziellen Moduls ausgeführt werden
Syntax:	<code><IfModule [!]Modulname Modulbezeichner> ... </IfModule></code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	All
Status:	Core
Modul:	core
Kompatibilität:	Modulbezeichner sind ab Version 2.1 verfügbar.

Der Container `<IfModule Test>...</IfModule>` wird dazu verwendet, Direktiven als abhängig von dem Vorhandensein eines speziellen Moduls zu kennzeichnen. Die Direktiven innerhalb eines `<IfModule>`-Abschnitts werden nur ausgeführt, wenn *Test* wahr ist. Ist *Test* falsch, wird alles zwischen der Start- und Endemarkierung ignoriert.

In der `<IfModule>`-Anweisung kann *Test* eine von zwei Formen annehmen:

- *Modul*
- *!Modul*

Im ersten Fall werden die Direktiven zwischen der Start- und Endemarkierung nur ausgeführt, das Modul namens *Modul* im Apache enthalten ist -- entweder einkompiliert oder mittels `LoadModule` dynamisch geladen. Die zweite Form dreht den Test um und führt die Direktiven nur aus, wenn *Modul* **nicht** enthalten ist.

Das Argument *Modul* kann entweder der Modulbezeichner oder der Dateiname des Moduls zum Zeitpunkt seiner Kompilierung sein. `rewrite_module` beispielsweise ist der Bezeichner und

`mod_rewrite.c` ist der Dateiname. Wenn ein Modul aus mehreren Quelltext-Dateien besteht, verwenden Sie den Namen der Datei, welche die Zeichenfolge `STANDARD20_MODULE_STUFF` enthält.

`<IfModule>`-Container können ineinander verschachtelt werden, um einfache Multi-Modul-Tests durchzuführen.

Dieser Container sollte verwendet werden, wenn Sie eine Konfigurationsdatei benötigen, die unabhängig davon funktioniert, ob ein bestimmtes Modul verfügbar ist oder nicht. Normalerweise ist es nicht notwendig, Direktiven in `<IfModule>`-Containern unterzubringen.



<u>Beschreibung:</u>	Fügt andere Konfigurationsdateien innerhalb der Server-Konfigurationsdatei ein
<u>Syntax:</u>	<code>Include <i>Dateiname</i> <i>Verzeichnis</i></code>
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Die Platzhalter-Suche ist verfügbar seit 2.0.41

Die Direktive erlaubt das Einfügen anderer Konfigurationsdateien in die Konfigurationsdatei des Servers.

Shell-typische (`fnmatch()`) Platzhalterzeichen können dazu verwendet werden, mehrere Dateien auf einmal in alphabetischer Reihenfolge einzufügen. Wenn `Include` darüber hinaus auf ein Verzeichnis anstatt auf eine Datei zeigt, liest der Apache alle Dateien in diesem Verzeichnis und allen Unterverzeichnissen ein. Das Einfügen ganzer Verzeichnisse ist jedoch nicht empfehlenswert, da temporäre Dateien sehr leicht versehentlich in einem Verzeichnis zurückgelassen werden, was [httpd](#) scheitern lassen kann.

Der angegebene Dateiname kann ein absoluter Pfad sein oder relativ zum `ServerRoot`-Verzeichnis angegeben werden.

Beispiele:

```
Include /usr/local/apache2/conf/ssl.conf
Include /usr/local/apache2/conf/vhosts/*.conf
```

Oder Sie geben Pfade relativ zu Ihrem `ServerRoot`-Verzeichnis an:

```
Include conf/ssl.conf
Include conf/vhosts/*.conf
```

Der Aufruf von `apachectl configtest` liefert eine Liste der Dateien, die während des Konfigurations-Tests verarbeitet werden:

```
root@host# apachectl configtest
Processing config file: /usr/local/apache2/conf/ssl.conf
Processing config file: /usr/local/apache2/conf/vhosts/vhost1.conf
Processing config file: /usr/local/apache2/conf/vhosts/vhost2.conf
Syntax OK
```

Siehe auch

- [apachectl](#)



Beschreibung:	Aktiviert persistente HTTP-Verbindungen
Syntax:	KeepAlive On Off
Voreinstellung:	KeepAlive On
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

Die Keep-Alive-Erweiterung von HTTP/1.0 und die HTTP/1.1-Funktionalität persistenter Verbindungen unterstützt langlebige HTTP-Sitzungen, die es erlauben, mehrere Anfragen über die gleich TCP-Verbindung zu senden. In einigen Fällen wurde eine Beschleunigung der Wartezeiten von beinahe 50% für HTML-Dokumente mit vielen Bildern festgestellt. Um Keep-Alive-Verbindungen zu aktivieren, setzen Sie `KeepAlive On`.

Bei HTTP/1.0-Clients werden Keep-Alive-Verbindungen nur dann verwendet, wenn sie vom Client eigens angefordert werden. Desweiteren können Keep-Alive-Verbindungen bei einem HTTP/1.0-Client nur dann verwendet werden, wenn die Länge des Inhalts im Voraus bekannt ist. Dies impliziert, dass dynamische Inhalte wie CGI-Ausgaben, SSI-Seiten und servergenerierte Verzeichnisaufstellungen im Allgemeinen keine Keep-Alive-Verbindungen mit HTTP/1.0-Clients verwenden. Bei HTTP/1.1-Clients sind Keep-Alive-Verbindungen Voreinstellung, solange nichts anderes angegeben ist. Wenn der Client es anfordert, wird Chunked-Encoding verwendet, um Inhalte mit unbekannter Länge über persistente Verbindungen zu senden.

Siehe auch

- [MaxKeepAliveRequests](#)



<u>Beschreibung:</u>	Zeitspanne, die der Server während persistenter Verbindungen auf nachfolgende Anfragen wartet
<u>Syntax:</u>	KeepAliveTimeout <i>Sekunden</i>
<u>Voreinstellung:</u>	KeepAliveTimeout 5
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core

Dies legt die Anzahl der Sekunden fest, die der Apache auf weitere Anfragen wartet, bevor er die Verbindung schließt. Nachdem einmal eine Anfrage entgegen genommen wurde, wird die durch die Direktive **Timeout** festgelegte Auszeit angewendet.

Auf stark belasteten Servern kann ein hoher **KeepAliveTimeout**-Wert zu Durchsatzminderungen führen. Je höher die Auszeit angegeben ist, desto länger ist der Apache damit beschäftigt, auf untätige Clients zu warten.



Beschreibung: Beschränkt die eingeschlossenen Zugriffskontrollen auf bestimmte HTTP-Methoden

Syntax: `<Limit Methode [Methode] ... > ...
</Limit>`

Kontext: Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess

AllowOverride: All

Status: Core

Modul: core

Zugriffskontrollen gelten normalerweise für **alle** Zugriffsmethoden, was normalerweise auch das gewünschte Verhalten ist. **Im Allgemeinen sollten Zugriffskontrollen nicht in einen `<Limit>`-Container gepackt werden.**

Der Sinn der Direktive `<Limit>` ist es, den Effekt der Zugriffskontrollen auf die angegebenen HTTP-Methoden zu beschränken. Bei allen anderen Methoden haben die in der `<Limit>`-Gruppe enthaltenen Zugriffsbeschränkungen **keine Wirkung**. Im folgenden Beispiel gilt die Zugriffskontrolle nur für die Methoden POST, PUT und DELETE. Alle anderen Methoden bleiben ungeschützt:

```
<Limit POST PUT DELETE>  
    Require valid-user  
</Limit>
```

Sie können eine oder mehrere der folgenden Methoden angeben: GET, POST, PUT, DELETE, CONNECT, OPTIONS, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK und UNLOCK. **Die Methodennamen unterscheiden zwischen Groß- und Kleinschreibung.** Wenn GET verwendet wird, sind HEAD-Anfragen ebenfalls eingeschränkt. Die TRACE-Methode kann nicht limitiert werden.

Wenn es um Zugriffsbeschränkungen geht, sollte ein `<LimitExcept>`-Container sollte immer einem `<Limit>`-Container vorgezogen werden, da `<LimitExcept>` einen Schutz gegen beliebige Methoden bietet.



Beschreibung:	Beschränkt Zugriffskontrollen auf alle HTTP-Methoden außer den genannten
Syntax:	<code><LimitExcept Methode [Methode] ... > ... </LimitExcept></code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	All
Status:	Core
Modul:	core

`<LimitExcept>` und `</LimitExcept>` werden dazu verwendet, eine Gruppe von Anweisungen zur Zugriffskontrolle zusammenzufassen, die dann auf jede HTTP-Methode angewendet werden, die **nicht** als Argument angegeben ist. D.h. dies ist das Gegenteil des `<Limit>`-Containers und kann zur Steuerung von Standard- und nicht-Standard-/unbekannten Methoden verwendet werden. Für weitere Einzelheiten lesen Sie bitte die Beschreibung zu `<Limit>`.

Beispiel:

```
<LimitExcept POST GET>
  Require valid-user
</LimitExcept>
```



<u>Beschreibung:</u>	Bestimmt die maximale Anzahl interner Umleitungen und verschachtelter Unteranfragen
<u>Syntax:</u>	<code>LimitInternalRecursion Zahl [Zahl]</code>
<u>Voreinstellung:</u>	<code>LimitInternalRecursion 10</code>
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Verfügbar ab Apache 2.0.47

Eine interne Umleitung erfolgt beispielsweise, wenn die Direktive [Action](#) verwendet wird, welche die Originalanfrage intern zu einem CGI-Skript weiterleitet. Eine Unteranfrage (*Anm.d.Ü.: engl. Subrequest*) ist ein Mechanismus des Apache, um herauszufinden, was bei einer URI geschehen würde, wäre sie angefordert worden. `mod_dir` z.B. verwendet Unteranfragen, um nach den Dateien zu suchen, die in der [DirectoryIndex](#)-Anweisung aufgeführt sind.

`LimitInternalRecursion` bewahrt den Server vor einem Absturz, wenn er in eine Endlosschleife aus internen Umleitungen oder Unteranfragen hineinläuft. Derartige Schleifen werden gewöhnlich durch Fehlkonfiguration verursacht.

Die Direktive setzt zwei verschiedene Begrenzungen, welche je Anfrage ausgewertet werden. Die erste *Zahl* bestimmt die maximale Anzahl der Umleitungen, die aufeinander folgen dürfen. Die zweite *Zahl* legt fest, wie tief Unteranfragen ineinander verschachtelt werden dürfen. Wenn Sie lediglich eine *Zahl* angeben, wird sie beiden Begrenzungen zugewiesen.

Beispiel

```
LimitInternalRecursion 5
```



Beschreibung:	Begrenzt die Gesamtgröße des vom Client gesendeten HTTP-Request-Body
Syntax:	<code>LimitRequestBody Bytes</code>
Voreinstellung:	<code>LimitRequestBody 0</code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	All
Status:	Core
Modul:	core

Die Direktive gibt die Anzahl der *Bytes* zwischen 0 (unbegrenzt) und 2147483647 (2GB) an, die im Request-Body (Datenteil der Anfrage) erlaubt sind.

Die Direktive `LimitRequestBody` erlaubt es dem Benutzer, die Größe des HTTP-Request-Bodys in dem Kontext zu begrenzen, in dem die Anweisung angegeben ist (Server, pro Verzeichnis, pro Datei oder pro Adresse). Wenn die Anfrage des Clients dieses Limit überschreitet, gibt der Server einen Fehler zurück anstatt die Anfrage zu bearbeiten. Die Größe des Datenteils einer Anfrage kann sehr stark variieren, abhängig von der Art der Ressource und den für diese Ressource erlaubten Methoden. CGI-Skripte verwenden den Datenteil üblicherweise zum Empfang von Formulardaten. Wird die PUT-Methode angewendet, dann muss der Wert mindestens so groß sein wie irgendeine Darstellungsform, die der Server für diese Ressource akzeptieren soll.

Die Direktive gibt dem Serveradministrator eine größere Kontrolle gegenüber abnormalem Verhalten von Clients, was bei der Vermeidung einiger Formen von Denial-of-Service-Attacken hilfreich sein kann.

Wenn Sie beispielsweise das Hochladen von Dateien zu einer bestimmten Adresse erlauben, aber die Größe der hochgeladenen Dateien auf 100K beschränken wollen, können Sie die folgende

Anweisung verwenden:

```
LimitRequestBody 102400
```



Beschreibung:	Begrenzt die Anzahl der HTTP-Request-Header, die vom Client entgegengenommen werden
Syntax:	<code>LimitRequestFields <i>Anzahl</i></code>
Voreinstellung:	<code>LimitRequestFields 100</code>
Kontext:	Serverkonfiguration
Status:	Core
Modul:	core

Anzahl ist ein Integer-Wert (eine positive Ganzzahl) zwischen 0 (unbegrenzt) und 32767. Die Voreinstellung wird durch die Konstante `DEFAULT_LIMIT_REQUEST_FIELDS` (100 bei der Auslieferung) zur Kompilierungszeit gesetzt.

Die Direktive `LimitRequestFields` erlaubt es dem Serveradministrator, die maximale Anzahl der in einem HTTP-Request erlaubten HTTP-Request-Header zu verändern. Für den Server muss dieser Wert größer sein als die Anzahl der Headerzeilen, die ein normaler Client senden könnte. Die Anzahl der Request-Header, die ein gewöhnlicher Client verwendet, überschreitet selten 20 Zeilen. Allerdings kann dies zwischen den verschiedenen Client-Ausführungen variieren, oft abhängig vom Ausmaß, mit dem der Anwender die genaue Content-Negotiation-Unterstützung seines Browsers konfiguriert hat. Optionale HTTP-Erweiterungen äußern sich oft in Form von HTTP-Headern.

Die Direktive gibt dem Serveradministrator eine größere Kontrolle gegenüber abnormalem Verhalten von Clients, was bei der Vermeidung einiger Formen von Denial-of-Service-Attacken hilfreich sein kann. Der Wert sollte erhöht werden, wenn normale Clients eine Fehlermeldung vom Server erhalten, die besagt, dass mit der Anfrage zu viele Headerzeilen gesendet wurden.

Beispiel:

LimitRequestFields 50



<u>Beschreibung:</u>	Begrenzt die Länge des vom Client gesendeten HTTP-Request-Headers
<u>Syntax:</u>	<code>LimitRequestFieldSize</code> <i>Bytes</i>
<u>Voreinstellung:</u>	<code>LimitRequestFieldSize</code> 8190
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive gibt die Anzahl der *Bytes* an, die in einem HTTP-Header erlaubt sind.

Die Direktive `LimitRequestFieldSize` erlaubt es dem Serveradministrator, die maximale Größe eines HTTP-Request-Headers zu verringern oder erhöhen. Für den Server muss der Wert groß genug sein, um eine beliebige Headerzeile einer normalen Client-Anfrage vorzuhalten. Die Größe variiert stark zwischen den verschiedenen Client-Ausführungen, oft abhängig vom Ausmaß, mit dem der Anwender die genaue Content-Negotiation-Unterstützung seines Browsers konfiguriert hat. SPNEGO-Authentisierungs-Header können bis zu 12392 Bytes lang sein.

Die Direktive gibt dem Serveradministrator eine größere Kontrolle gegenüber abnormalem Verhalten von Clients, was bei der Vermeidung einiger Formen von Denial-of-Service-Attacks hilfreich sein kann.

Beispiel:

```
LimitRequestFieldSize 4094
```

Unter normalen Umständen sollte die Voreinstellung nicht verändert werden.



Beschreibung:	Begrenzt die Länge der vom Client entgegengenommenen HTTP-Anfragezeile
Syntax:	<code>LimitRequestLine Bytes</code>
Voreinstellung:	<code>LimitRequestLine 8190</code>
Kontext:	Serverkonfiguration
Status:	Core
Modul:	core

Die Direktive legt die Anzahl der *Bytes* fest, die in der HTTP-Anfragezeile erlaubt sind.

Die Direktive `LimitRequestLine` erlaubt es dem Serveradministrator, die maximale Größe der HTTP-Anfragezeile zu verringern oder erhöhen. Da die Anfragezeile aus der HTTP-Methode, der URI und der Protokollversion besteht, bedeutet die `LimitRequestLine`-Direktive eine Beschränkung der Länge der für eine Anfrage an den Server erlaubten Anfrage-URI. Für den Server muss der Wert groß genug sein, um jeden seiner Ressourcennamen vorzuhalten, einschließlich aller Informationen, die im Query-String einer GET-Anfrage übergeben werden können.

Die Direktive gibt dem Serveradministrator eine größere Kontrolle gegenüber abnormalem Verhalten von Clients, was bei der Vermeidung einiger Formen von Denial-of-Service-Attacken hilfreich sein kann.

Beispiel:

```
LimitRequestLine 4094
```

Unter normalen Umständen sollte die Voreinstellung nicht verändert werden.



Beschreibung:	Begrenzt die Größe eines XML-basierten Request-Bodys
Syntax:	LimitXMLRequestBody <i>Bytes</i>
Voreinstellung:	LimitXMLRequestBody 1000000
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	All
Status:	Core
Modul:	core

Dies gibt die Grenze für die maximale Größe (in Bytes) des XML-basierten Request-Bodys an. Der Wert 0 deaktiviert diese Prüfung.

Beispiel:

```
LimitXMLRequestBody 0
```



Beschreibung:	Wendet die enthaltenen Direktiven nur auf die entsprechenden URLs an
Syntax:	<code><Location URL-Pfad URL> ... </Location></code>
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

Die Direktive `<Location>` begrenzt die Reichweite der enthaltenen Anweisungen auf URLs. Sie ist der Direktive `<Directory>` ähnlich und startet einen Abschnitt, der mit der Anweisung `</Location>` abgeschlossen wird. `<Location>`-Container werden, nachdem die `<Directory>`-Container und `.htaccess`-Dateien gelesen wurden, und nach den `<Files>`-Containern, in der Reihenfolge ausgeführt, in der sie in der Konfigurationsdatei erscheinen.

`<Location>`-Abschnitte operieren vollständig außerhalb des Dateisystems. Dies hat mehrere Konsequenzen. Am Wichtigsten, `<Location>`-Anweisungen sollten nicht dafür verwendet werden, den Zugriff zu Teilen des Dateisystems zu steuern. Da mehrere unterschiedliche URLs auf die gleiche Stelle des Dateisystems zeigen können, könnte eine solche Zugriffskontrolle u.U. umgangen werden.

Wann sollte `<Location>` verwendet werden

Verwenden Sie `<Location>`, um Anweisungen auf Inhalte anzuwenden, die außerhalb des Dateisystems abgelegt sind. Benutzen Sie `<Directory>` und `<Files>` für Inhalte, die innerhalb des Dateisystems abgelegt sind. Eine Ausnahme bildet `<Location />`, welches ein einfacher Weg ist, um eine Konfiguration auf den gesamten Server anzuwenden.

Für alle nicht-Proxy-Anfragen ist die entsprechende URL ein URL-Pfad in

der Form `/path/`. Es dürfen weder ein Schema, noch ein Hostname, noch ein Port, noch ein Query-String einbezogen werden. Für Proxy-Anfragen hat die Vergleichs-URL die Form `schema://servername/path`. Das Präfix muss angegeben werden.

Die URL kann Platzhalter verwenden. In einer Zeichenfolge mit Platzhaltern entspricht `?` einem einzelnen Zeichen und `*` einer beliebigen Zeichenfolge.

Erweiterte reguläre Ausdrücke können ebenfalls verwendet werden, indem das Zeichen `~` hinzugefügt wird. Beispielsweise würde

```
<Location ~ "/(extra|special)/data">
```

auf URLs passen, welche die Zeichenfolge `/extra/data` oder `/special/data` enthalten. Die Direktive `<LocationMatch>` verhält sich genauso wie `<Location>` mit regulären Ausdrücken.

Die Funktionalität von `<Location>` ist insbesondere dann nützlich, wenn sie mit der `SetHandler`-Direktive kombiniert wird. Um zum Beispiel Statusabfragen zu aktivieren, sie aber nur von Browsern aus `foo.com` zuzulassen, könnten Sie schreiben:

```
<Location /status>
  SetHandler server-status
  Order Deny,Allow
  Deny from all
  Allow from .foo.com
</Location>
```

Anmerkung zu / (Schrägstrich, Slash)

Das Slash-Zeichen hat eine besondere Bedeutung, je nachdem, wo es in der URL erscheint. Manche werden sein Verhalten vom Dateisystem gewohnt sein, wo mehrere aufeinanderfolgende Schrägstriche häufig zu einem Schrägstrich zusammengefaßt werden (*d.h.* `/home///foo`

ist das gleiche wie /home/foo). Im URL-Raum ist dies nicht notwendigerweise genauso. Bei der Direktive `<LocationMatch>` und der `<Location>`-Version mit regulären Ausdrücken müssen Sie explizit mehrere Schrägstriche angeben, wenn Sie genau dies beabsichtigen.

Beispielsweise würde `<LocationMatch ^/abc>` auf die angeforderte URL /abc passen, nicht aber auf //abc. Die Direktive `<Location>` (ohne reguläre Ausdrücke) verhält sich ähnlich, wenn sie für Proxy-Anfragen verwendet wird. Wenn `<Location>` (ohne reguläre Ausdrücke) jedoch für nicht-Proxy-Anfragen verwendet wird, werden stillschiegend mehrere Schrägstriche mit einem einzigen Schrägstrich gleichgesetzt. Geben Sie beispielsweise `<Location /abc/def>` an und die Anfrage lautet auf /abc//def, dann greift die Anweisung.

Siehe auch

- [Wie die Abschnitte <Directory>, <Location> und <Files> arbeiten](#) für eine Erläuterung, wie diese verschiedenen Abschnitte miteinander kombiniert werden, wenn eine Anfrage empfangen wird



Beschreibung: Wendet die enthaltenen Direktiven nur auf URLs an, die auf reguläre Ausdrücke passen

Syntax: `<LocationMatch regex> ...
</LocationMatch>`

Kontext: Serverkonfiguration, Virtual Host

Status: Core

Modul: core

Die Direktive `<LocationMatch>` begrenzt die Reichweite der enthaltenen Anweisungen in der gleichen Weise wie `<Location>` auf URLs. Sie verwendet jedoch reguläre Ausdrücke als Argument anstelle einer einfachen Zeichenkette. Beispielsweise würde

```
<LocationMatch "/(extra|special)/data">
```

auf URLs passen, welche die Zeichenfolge `/extra/data` oder `/special/data` enthalten.

Siehe auch

- [Wie die Abschnitte `<Directory>`, `<Location>` und `<Files>` arbeiten](#) für eine Erläuterung, wie diese verschiedenen Abschnitte miteinander kombiniert werden, wenn eine Anfrage empfangen wird



Beschreibung:	Steuert die Ausführlichkeit des Fehlerprotokolls
Syntax:	LogLevel <i>Level</i>
Voreinstellung:	LogLevel warn
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

LogLevel stellt die Ausführlichkeit der Nachrichten ein, die im Fehlerprotokoll aufgezeichnet werden (siehe Direktive [ErrorLog](#)). Die folgenden, nach absteigender Aussagekraft sortierten *Level* sind verfügbar:

Level	Beschreibung	Beispiel
emerg	Notfall - das System ist unbenutzbar.	"Child cannot open lock file. Exiting" (<i>Anm.d.Ü.:</i> "Kindprozess kann die Lock-Datei nicht öffnen. Beende Programm")
alert	Maßnahmen müssen unverzüglich ergriffen werden.	"getpwuid: couldn't determine user name from uid" (<i>Anm.d.Ü.:</i> "getpwuid: kann keinen Benutzernamen aus der UID ermitteln")
crit	Kritischer Zustand.	"socket: Failed to get a socket, exiting child" (<i>Anm.d.Ü.:</i> "socket: Socket-Zuweisung fehlgeschlagen, beende Kindprozess")
error	Fehlerbedingung.	"Premature end of script headers" (<i>Anm.d.Ü.:</i> "Vorzeitiges Ende der Skript-Header")
warn	Warnung.	"child process 1234 did not exit, sending another SIGHUP" (<i>Anm.d.Ü.:</i> "Kindprozess 1234 nicht beendet, sende ein weiteres SIGHUP")
notice	Normaler, aber	"httpd: caught SIGBUS, attempting to dump

	signifikanter Zustand.	core in ..." (Anm.d.Ü.: "httpd: SIGBUS empfangen, versuche Speicherabbild nach ... zu schreiben")
info	Information.	"Server seems busy, (you may need to increase StartServers, or Min/MaxSpareServers)..." (Anm.d.Ü.: "Server scheint beschäftigt zu sein, (möglicherweise müssen Sie StartServers oder Min/MaxSpareServers erhöhen)")
debug	Debug-Level-Nachrichten	"Opening config file ..." (Anm.d.Ü.: "Öffne Konfigurationsdatei ...")

Geben Sie einen bestimmten Level an, denn werden Nachrichten von allen höheren Leveln ebenso angezeigt. Z.B.: Wenn `LogLevel info` eingestellt ist, dann werden Nachrichten der Log-Level `notice` und `warn` ebenso eingetragen.

Es wird empfohlen, mindestens den Level `crit` zu verwenden.

Beispiel:

```
LogLevel notice
```

Hinweis

Beim Protokollieren in eine reguläre Datei können Nachrichten des Levels `notice` nicht unterdrückt werden und werden daher immer protokolliert. Dies trifft allerdings nicht zu, wenn mittels `syslog` protokolliert wird.



<u>Beschreibung:</u>	Anzahl der Anfragen, die bei einer persistenten Verbindung zulässig sind
<u>Syntax:</u>	MaxKeepAliveRequests <i>Anzahl</i>
<u>Voreinstellung:</u>	MaxKeepAliveRequests 100
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive `MaxKeepAliveRequests` begrenzt die Anzahl der Anfragen, die pro Verbindung zulässig sind, wenn `KeepAlive` eingeschaltet ist. Bei der Einstellung 0 sind unbegrenzt viele Anfragen erlaubt. Wir empfehlen für diese Einstellung einen hohen Wert für eine maximale Serverleistung.

Beispiel:

```
MaxKeepAliveRequests 500
```



<u>Beschreibung:</u>	Number of ranges allowed before returning the complete resource
<u>Syntax:</u>	MaxRanges default unlimited none <i>number-of-ranges</i>
<u>Voreinstellung:</u>	MaxRanges 200
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Available in Apache HTTP Server 2.2.21 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



MergeTrailers Direktive

<u>Beschreibung:</u>	Determines whether trailers are merged into headers
<u>Syntax:</u>	MergeTrailers [on off]
<u>Voreinstellung:</u>	MergeTrailers off
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	2.2.28 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



Beschreibung:	Bestimmt eine IP-Adresse für den Betrieb namensbasierter virtueller Hosts
Syntax:	<code>NameVirtualHost Adresse[:Port]</code>
Kontext:	Serverkonfiguration
Status:	Core
Modul:	core

Die Direktive `NameVirtualHost` ist erforderlich, wenn Sie [namensbasierte virtuelle Hosts](#) konfigurieren möchten.

Obwohl *Adresse* eine Hostname sein kann, wird empfohlen, dass Sie stets eine IP-Adresse verwenden, z.B.:

```
NameVirtualHost 111.22.33.44
```

Mit der `NameVirtualHost`-Anweisung geben Sie die IP-Adresse an, unter der der Server Anfragen für namensbasierte virtuelle Hosts entgegennimmt. Das ist üblicherweise die Adresse, zu der die Namen Ihrer namensbasierten virtuellen Hosts aufgelöst werden. Falls eine Firewall oder ein anderer Proxy die Anfrage in Empfang nimmt und Sie zu einer weiteren IP-Adresse des Servers weiterleitet, müssen Sie die IP-Adresse der physikalischen Schnittstelle der Maschine angeben, welche die Anfragen bedient. Wenn Sie mehrere namensbasierte Hosts an verschiedenen Adressen betreiben, wiederholen Sie einfach die Anweisung für jede Adresse.

Anmerkung

Beachten Sie, dass der "Hauptserver" und jeder `_default_`-Server **niemals** bei einer Anfrage an einer `NameVirtualHost`-IP-Adresse bedient wird (es sei denn, Sie geben aus irgendwelchen Gründen `NameVirtualHost` an, definieren dann aber keine `VirtualHosts` für diese Adresse).

Optional können Sie die Nummer eines Ports angeben, an dem namensbasierte virtuelle Hosts verwendet werden sollen. Beispiel:

```
NameVirtualHost 111.22.33.44:8080
```

IPv6-Adressen müssen, wie im folgenden Beispiel angegeben, in eckige Klammern eingeschlossen werden:

```
NameVirtualHost [2001:db8::a00:20ff:fea7:ccea]:8080
```

Um an allen Schnittstellen Anfragen zu empfangen, können Sie `*` als Argument verwenden.

```
NameVirtualHost *
```

Argument der Direktive `<VirtualHost>`

Beachten Sie, dass das Argument der `<VirtualHost>`-Anweisung exakt auf das Argument der `NameVirtualHost`-Anweisung passen muss.

```
NameVirtualHost 1.2.3.4
<VirtualHost 1.2.3.4>
# ...
</VirtualHost>
```

Siehe auch

- [Dokumentation zu virtuellen Hosts](#)



Beschreibung:	Definiert, welche Eigenschaften oder Funktionen in einem bestimmten Verzeichnis verfügbar sind
Syntax:	Options [+ -] <i>Option</i> [[+ -] <i>Option</i>] ...
Voreinstellung:	Options All
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	Options
Status:	Core
Modul:	core

Die Direktive `Options` steuert, welche Eigenschaften bzw. Funktionen in einem bestimmten Verzeichnis verfügbar sind.

Option kann auf None gesetzt werden, wobei keine der besonderen Eigenschaften verfügbar sind, oder auf eines oder mehrere der folgenden:

All

Alle Optionen außer `MultiViews`. Dies ist die Voreinstellung.

ExecCGI

Die Ausführung von CGI-Skripten, welche `mod_cgi` verwenden, ist erlaubt.

FollowSymLinks

Der Server folgt symbolischen Links in diesem Verzeichnis.

Auch wenn der Server symbolischen Links folgt, bedeutet dies *nicht*, dass der zum Abgleich gegen `<Directory>`-Abschnitte verwendete Pfadname wechselt.

Beachten Sie auch, dass diese Option innerhalb eines `<Location>`-Abschnitts **ignoriert wird**.

Includes

Server Side Includes, die von [mod_include](#) bereitgestellt werden, sind erlaubt.

IncludesNOEXEC

Server Side Includes sind erlaubt, `#exec cmd` und `#exec cgi` sind jedoch deaktiviert. Es ist aber noch möglich, CGI-Skripte aus [ScriptAlias](#)-Verzeichnissen mittels `#include virtual` einzubinden.

Indexes

Wenn eine URL, die auf ein Verzeichnis zeigt, in dem sich keine durch [DirectoryIndex](#) definierte Indexdatei (z.B. `index.html`) befindet, dann liefert [mod_autoindex](#) eine formatierte Auflistung des Verzeichnisses zurück.

MultiViews

"MultiViews" sind bei der Verwendung von [mod_negotiation](#) erlaubt (siehe [Content-Negotiation](#)).

SymLinksIfOwnerMatch

Der Server folgt nur symbolischen Links, bei denen die Zieldatei bzw. das Zielverzeichnis der gleichen Benutzererkennung gehört, wie der Link.

Anmerkung

Diese Option wird innerhalb eines [<Location>](#)-Abschnitts ignoriert.

Wenn mehrere [Options](#) auf ein Verzeichnis angewandt werden können, dann wird normalerweise die spezifischste (*Anm.d.Ü.: Gemeint ist die zuletzt ausgeführte Option.*) verwendet und alle anderen werden ignoriert; die Optionen werden nicht vermischt. (Siehe auch [Wie Abschnitte zusammengeführt werden.](#)) Wenn jedoch *allen* Optionen der [Options](#)-Anweisung eines der Zeichen + oder - vorangestellt wird,

werden die Optionen zusammengemischt. Jede Option mit vorangestelltem + wird zu den momentan gültigen Optionen hinzugefügt und jede Option mit vorangestelltem - wird aus den derzeit gültigen Optionen entfernt.

So wird zum Beispiel ohne die Zeichen + und -

```
<Directory /web/docs>
  Options Indexes FollowSymLinks
</Directory>

<Directory /web/docs/spec>
  Options Includes
</Directory>
```

für das Verzeichnis /web/docs/spec wird jetzt lediglich Includes gesetzt. Wenn die zweite Options-Anweisung jedoch +- und --Zeichen verwenden würde,

```
<Directory /web/docs>
  Options Indexes FollowSymLinks
</Directory>

<Directory /web/docs/spec>
  Options +Includes -Indexes
</Directory>
```

dann würden die Optionen FollowSymLinks und Includes für das Verzeichnis /web/docs/spec gesetzt.

Anmerkung

Die Verwendung von -IncludesNOEXEC oder -Includes deaktiviert Server Side Includes unabhängig von der vorigen Einstellung vollständig.

Die Voreinstellung ist All, sofern keine anderen Angaben gemacht wurden.



<u>Beschreibung:</u>	Protocol for a listening socket
<u>Syntax:</u>	Protocol <i>protocol</i>
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Available in Apache 2.1.5 and later. On Windows, from Apache 2.3.3 and later.

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.

Siehe auch

- [AcceptFilter](#)
- [Listen](#)



Beschreibung:	Register non-standard HTTP methods
Syntax:	RegisterHttpMethod <i>method</i> [<i>method</i> [...]]
Kontext:	Serverkonfiguration
Status:	Core
Modul:	core

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



Beschreibung:	Wählt die authentisierten Benutzer aus, die auf eine Ressource zugreifen können
Syntax:	<code>Require Name [Name] ...</code>
Kontext:	Verzeichnis, .htaccess
AllowOverride:	AuthConfig
Status:	Core
Modul:	core

Die Direktive wählt aus, welche authentisierten Benutzer auf eine Ressource zugreifen dürfen. Folgende Syntax ist erlaubt:

Require user *User-ID* [*User-ID*] ...

Nur die genannten Benutzer dürfen auf die Ressource zugreifen.

Require group *Gruppenname* [*Gruppenname*] ...

Nur Benutzer der genannten Gruppen dürfen auf die Ressource zugreifen.

Require valid-user

Alle gültigen Benutzer dürfen auf die Ressource zugreifen.

Require muss von den Direktiven [AuthName](#) und [AuthType](#) sowie Direktiven wie [AuthUserFile](#) und [AuthGroupFile](#) (zur Definition von Benutzern und Gruppen) begleitet werden, um korrekt zu funktionieren.
Beispiel:

```
AuthType Basic
AuthName "Geschützte Ressource"
AuthUserFile /web/users
AuthGroupFile /web/groups
Require group admin
```

Zugriffskontrollen, die in dieser Form angewandt werden, gelten für **alle** Methoden. **Dies ist normalerweise gewünscht.** Wenn Sie Zugriffskontrollen nur auf bestimmte Methoden anwenden möchten,

während andere Methoden ungeschützt bleiben, dann müssen Sie die **Require**-Anweisung innerhalb eines `<Limit>`-Abschnitts platzieren.

Siehe auch

- [Satisfy](#)
- [mod_authz_host](#)



<u>Beschreibung:</u>	Begrenzt den CPU-Verbrauch von Prozessen, die von Apache-Kindprozessen gestartet wurden
<u>Syntax:</u>	<code>RLimitCPU Sekunden max [Sekunden max]</code>
<u>Voreinstellung:</u>	unbestimmt; verwendet die Voreinstellung des Systems
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	All
<u>Status:</u>	Core
<u>Modul:</u>	core

Akzeptiert einen oder zwei Parameter. Der erste Parameter setzt eine weiche Ressourcenbegrenzung für alle Prozesse, der zweite Parameter setzt die Maximalgrenze für die Ressourcennutzung. Jeder der Parameter kann eine Zahl oder max sein. max zeigt dem Server an, dass das vom Betriebssystem erlaubte Maximum verwendet werden soll. Das Anheben der maximal erlaubten Ressourcennutzung erfordert, dass der Server als root läuft, zumindest in der anfänglichen Startphase.

Dies wird auf Prozesse angewendet, die von Anfragen bearbeitenden Apache-Kindprozessen abgespalten werden, nicht auf die Apache-Kindprozesse selbst. Das beinhaltet CGI-Skripte und SSI-exec-Befehle, nicht jedoch Prozesse, die vom Apache-Elternprozess abgespalten werden, wie z.B. Protokollierung.

CPU-Ressourcenbegrenzung wird in Sekunden pro Prozess ausgedrückt.

Siehe auch

- [RLimitMEM](#)
- [RLimitNPROC](#)



<u>Beschreibung:</u>	Begrenzt den Speicherverbrauch von Prozessen, die von Apache-Kindprozessen gestartet wurden
<u>Syntax:</u>	<code>RLimitMEM Bytes max [Bytes max]</code>
<u>Voreinstellung:</u>	unbestimmt; verwendet die Voreinstellung des Systems
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	All
<u>Status:</u>	Core
<u>Modul:</u>	core

Akzeptiert einen oder zwei Parameter. Der erste Parameter setzt eine weiche Ressourcenbegrenzung für alle Prozesse, der zweite Parameter setzt die Maximalgrenze für die Ressourcennutzung. Jeder der Parameter kann eine Zahl oder max sein. max zeigt dem Server an, dass das vom Betriebssystem erlaubte Maximum verwendet werden soll. Das Anheben der maximal erlaubten Ressourcennutzung erfordert, dass der Server als root läuft, zumindest in der anfänglichen Startphase.

Dies wird auf Prozesse angewendet, die von Anfragen bearbeitenden Apache-Kindprozessen abgespalten werden, nicht auf die Apache-Kindprozesse selbst. Das beinhaltet CGI-Skripte und SSI-exec-Befehle, nicht jedoch Prozesse, die vom Apache-Elternprozess abgespalten werden, wie z.B. Protokollierung.

Die Begrenzung des Speicherverbrauchs wird in Bytes pro Prozess ausgedrückt.

Siehe auch

- [RLimitCPU](#)
- [RLimitNPROC](#)



<u>Beschreibung:</u>	Begrenzt die Anzahl der Prozesse, die von Prozessen gestartet werden können, die ihrerseits von Apache-Kinprozessen gestartet wurden
<u>Syntax:</u>	<code>RLimitNPROC Zahl max [Zahl max]</code>
<u>Voreinstellung:</u>	unbestimmt; verwendet die Voreinstellung des Systems
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	All
<u>Status:</u>	Core
<u>Modul:</u>	core

Akzeptiert einen oder zwei Parameter. Der erste Parameter setzt eine weiche Ressourcenbegrenzung für alle Prozesse, der zweite Parameter setzt die Maximalgrenze für die Ressourcennutzung. Jeder der Parameter kann eine Zahl oder max sein. max zeigt dem Server an, dass das vom Betriebssystem erlaubte Maximum verwendet werden soll. Das Anheben der maximal erlaubten Ressourcennutzung erfordert, dass der Server als root läuft, zumindest in der anfänglichen Startphase.

Dies wird auf Prozesse angewendet, die von Anfragen bearbeitenden Apache-Kindprozessen abgespalten werden, nicht auf die Apache-Kindprozesse selbst. Dies beinhaltet CGI-Skripte und SSI-exec-Befehle, nicht jedoch Prozesse, die vom Apache-Elternprozess abgespalten werden, wie z.B. Protokollierung.

Prozessbegrenzungen steuern die Anzahl der Prozesse pro Benutzer.

Anmerkung

Wenn CGI-Prozesse nicht unter anderen Benutzerkennungen als der User-ID des Webservers laufen, dann beschränkt diese Direktive die Anzahl der Prozesse, die der Server selbst erstellen kann.

Kennzeichen einer solchen Situation sind **cannot fork**-Meldungen
(Anm.d.Ü.: kann nicht abspalten) in der Datei `error_log`.

Siehe auch

- [RLimitMEM](#)
- [RLimitCPU](#)



Beschreibung:	Zusammenspiel von rechnerbasierter Zugriffskontrolle und Benutzerauthentisierung
Syntax:	Satisfy Any All
Voreinstellung:	Satisfy All
Kontext:	Verzeichnis, .htaccess
AllowOverride:	AuthConfig
Status:	Core
Modul:	core
Kompatibilität:	Wird seit Version 2.0.51 von <code><Limit></code> und <code><LimitExcept></code> beeinflusst

Verfahrensweise für den Zugriff, falls sowohl `Allow` als auch `Require` verwendet werden. Der Parameter kann entweder `All` oder `Any` sein. Die Direktive ist nur dann nützlich, wenn der Zugriff zu einem bestimmten Bereich durch Benutzername/Passwort *und* Clientrechner-Adressen eingeschränkt ist. In diesem Fall verlangt die Voreinstellung (`All`), dass der Client die Adressbeschränkung passiert *und* eine gültige Benutzerkennung und ein gültiges Passwort übermittelt. Mit der Auswahl `Any` wird dem Client der Zugriff erlaubt, wenn er entweder die Rechner-Beschränkung passiert oder einen gültigen Benutzernamen und ein gültiges Passwort übermittelt. Dies kann verwendet werden, um einen Bereich mit einem Passwort zu schützen, jedoch Clients von bestimmten Adressen ohne Abfrage des Passwortes zuzulassen.

Wenn Sie beispielsweise möchten, dass Personen aus Ihrem privaten Netzwerk unbeschränkten Zugriff zu Teilen Ihres Webangebots haben, jedoch verlangen, dass Personen außerhalb Ihres privaten Netzwerks ein Passwort übergeben müssen, können Sie eine Konfiguration ähnlich der folgenden verwenden:

```
Require valid-user
Allow from 192.168.1
Satisfy Any
```

Seit Version 2.0.51 können **Satisfy**-Anweisungen durch [<Limit>](#)- und [<LimitExcept>](#)-Abschnitte auf bestimmte Methoden beschränkt werden.

Siehe auch

- [Allow](#)
- [Require](#)



<u>Beschreibung:</u>	Methode zur Ermittlung des Interpreters von CGI-Skripten
<u>Syntax:</u>	ScriptInterpreterSource Registry Registry-Strict Script
<u>Voreinstellung:</u>	ScriptInterpreterSource Script
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	ausschließlich Win32; Die Option Registry-Strict ist verfügbar seit Apache 2.0.

Die Direktive steuert, wie der Apache den Interpreter zur Ausführung von CGI-Skripten bestimmt. Die Voreinstellung ist `Script`. Dies veranlaßt den Apache, den Interpreter zu verwenden, auf den die Shebang-Zeile (erste Zeile, beginnt mit `#!`) im Skript zeigt. Auf Win32-Systemen sieht diese Zeile üblicherweise so aus:

```
#!C:/Perl/bin/perl.exe
```

oder, wenn `perl` im Pfad (Umgebungsvariable `PATH`) liegt, einfach:

```
#!perl
```

Die Einstellung `ScriptInterpreterSource Registry` veranlaßt eine Suche in `HKEY_CLASSES_ROOT` der Windows-Registrierungsdatenbank und verwendet die Endung der Skript-Datei (z.B. `.pl`) als Suchargument. Der durch den Unterschlüssel `Shell\ExecCGI\Command` oder, falls dieser nicht existiert, `Shell\Open\Command` definierte Befehl wird zum Öffnen der Skript-

Datei verwendet. Wenn der Schlüssel zur Dateieindung oder beide Unterschlüssel fehlen, dann verwendet der Apache die Option `Script`.

Sicherheit

Seien Sie vorsichtig, `ScriptInterpreterSource Registry` bei Verzeichnissen zu verwenden, auf die eine `ScriptAlias`-Anweisung zeigt, denn der Apache versucht **jede** Datei innerhalb des Verzeichnisses auszuführen. Die Einstellung `Registry` kann unerwünschte Programmaufrufe bei Dateien verursachen, die üblicherweise nicht ausgeführt werden. Auf den meisten Windows-Systemen beispielsweise startet der voreingestellte Öffnen-Befehl für `.htm`-Dateien den Microsoft Internet Explorer, so dass jede HTTP-Anfrage nach einer existierenden `.htm`-Datei im Skript-Verzeichnis den Browser im Hintergrund starten würde. Dies ist eine wirksame Methode, Ihr System binnen etwa einer Minute zum Absturz zu bringen.

Die seit Apache 2.0 neue Option `Registry-Strict` macht das gleiche wie `Registry`, verwendet jedoch nur den Unterschlüssel `Shell\ExecCGI\Command`. Der Schlüssel `ExecCGI` ist gewöhnlich nicht voreingestellt. Er muss manuell eingerichtet werden und schützt Ihr System so for versehentlichen Programmaufrufen.



Beschreibung:	E-Mail-Adresse, die der Server in Fehlermeldungen einfügt, welche an den Client gesendet werden
Syntax:	ServerAdmin <i>E-Mail-Adresse</i> URL
Kontext:	Serverkonfiguration, Virtual Host
Status:	Core
Modul:	core

ServerAdmin legt die Kontaktadresse fest, die der Server in jede Fehlermeldung einfügt, die er an den Client zurückschickt. Wenn `httpd` das übergebene Argument nicht als URL erkennt, nimmt er an, dass es sich um eine *E-Mail-Adresse* handelt und stellt in Hyperlinks `mailto:` voran. Es ist jedoch sogar sinnvoll, eine E-Mail-Adresse zu verwenden, da viele CGI-Skripte davon ausgehen. Wenn Sie eine URL verwenden möchten, sollten Sie auf einem anderen unter Ihrer Kontrolle stehenden Server verweisen. Andernfalls können Besucher Sie im Fehlerfall möglicherweise nicht kontaktieren.

Es kann sich lohnen, hierfür eine reservierte Adresse anzugeben, z.B.

```
ServerAdmin www-admin@foo.example.com
```

da Anwender nicht unbedingt erwähnen, dass sie vom Server sprechen!



<u>Beschreibung:</u>	Alternativer Name für einen Host, der verwendet wird, wenn Anfragen einem namensbasierten virtuellen Host zugeordnet werden
<u>Syntax:</u>	<code>ServerAlias Hostname [Hostname] ...</code>
<u>Kontext:</u>	Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive `ServerAlias` bestimmt die alternativen Namen eines Hosts zur Verwendung mit [namensbasierten virtuellen Hosts](#).

```
<VirtualHost *>
ServerName server.domain.com
ServerAlias server server2.domain.com server2
# ...
</VirtualHost>
```

Siehe auch

- [Apache-Dokumentation zu virtuellen Hosts](#)



<u>Beschreibung:</u>	Rechnername und Port, die der Server dazu verwendet, sich selbst zu identifizieren
<u>Syntax:</u>	<code>ServerName voll-qualifizierter-Domainname[:port]</code>
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Diese Direktive löst in Version 2.0 die Funktionalität der Direktive <code>Port</code> aus Version 1.3 ab.

Die Direktive `ServerName` bestimmt den Rechnernamen und Port, den der Server dazu verwendet, sich selbst zu identifizieren. Diese werden bei der Erstellung von Umleitungs-URLs benötigt. Wenn beispielsweise der Name der Maschine, die den Webserver beherbergt, `simple.example.com` lautet, die Maschine jedoch auch einen DNS-Alias `www.example.com` besitzt und Sie den Webserver so identifizieren möchten, sollten Sie die folgende Anweisung verwenden:

```
ServerName www.example.com:80
```

Wenn kein `ServerName` angegeben wurde, dann versucht der Server den Rechnernamen mittels eines Reverse-Lookup herzuleiten. Wenn kein Port in der `ServerName`-Anweisung angegeben wurde, dann verwendet der Server den Port der eingegangenen Anfrage. Für eine optimale Zuverlässigkeit und Berechenbarkeit sollten Sie einen eindeutigen Rechnernamen und Port angeben, in dem Sie die Direktive `ServerName` verwenden.

Wenn Sie [namensbasierte virtuelle Hosts](#) verwenden, gibt `ServerName` innerhalb eines `<VirtualHost>`-Abschnitts an, welcher Hostname im `Host : -Header` der Anfrage auftauchen muss, damit sie diesem virtuellen Host zugeordnet wird.

Lesen Sie bitte die Beschreibung der Direktive [UseCanonicalName](#) für Einstellungen, die bestimmen, ob selbstreferenzierende URLs (z.B. vom Modul [mod_dir](#)) auf den angegebenen Port zeigen oder auf die Portnummern die in der Anfrage des Clients angegeben ist.

Siehe auch

- [Probleme bezüglich DNS und Apache](#)
- [Apache-Dokumentation zu virtuellen Hosts](#)
- [UseCanonicalName](#)
- [NameVirtualHost](#)
- [ServerAlias](#)



<u>Beschreibung:</u>	Veralteter URL-Pfad für einen namensbasierten virtuellen Host, auf den von einem inkompatiblen Browser zugegriffen wird
<u>Syntax:</u>	ServerPath <i>URL - Pfad</i>
<u>Kontext:</u>	Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive `ServerPath` legt den veralteten (*Anm.d.Ü.: Gemeint ist eigentlich "Altlast" aufgrund antiquierter Clients.*) URL-Pfad eines Hosts zur Verwendung mit [namensbasierten virtuellen Hosts](#) fest.

Siehe auch

- [Apache-Dokumentation zu virtuellen Hosts](#)



<u>Beschreibung:</u>	Basisverzeichnis der Serverinstallation
<u>Syntax:</u>	ServerRoot <i>Verzeichnis</i>
<u>Voreinstellung:</u>	ServerRoot /usr/local/apache
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive **ServerRoot** bestimmt das Verzeichnis, in dem der Server installiert ist. Üblicherweise enthält es die Unterverzeichnisse `conf/` und `logs/`. Relative Pfadangaben anderer Direktiven (wie z.B. **Include** oder **LoadModule**) werden relativ zu diesem Verzeichnis betrachtet.

Beispiel

```
ServerRoot /home/httpd
```

Siehe auch

- [Die httpd-Option -d](#)
- [Sicherheitshinweise](#) für Informationen, wie die Rechte auf das **ServerRoot**-Verzeichnis richtig gesetzt werden



Beschreibung:	Konfiguriert die Fußzeile von servergenerierten Dokumenten
Syntax:	ServerSignature On Off EMail
Voreinstellung:	ServerSignature Off
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	All
Status:	Core
Modul:	core

Die Direktive [ServerSignature](#) ermöglicht die Gestaltung einer unter servergenerierten Dokumenten (z.B. Fehlerdokumente, FTP-Verzeichnislisten von [mod_proxy](#), [mod_info](#)-Ausgaben, ...) angefügten Fußzeile. Ein möglicher Grund für die Aktivierung einer solchen Fußzeile ist, dass der Anwender bei einer Kette von Proxy-Servern oft keine Möglichkeit hat, zu erkennen, welcher der verketteten Server gegenwärtig die zurückgegebene Fehlermeldung produziert hat.

Die (Vor-)Einstellung `Off` unterdrückt die Fußzeile (und ist damit kompatibel zum Verhalten des Apache 1.2 und früher). Die Einstellung `On` fügt schlicht eine Zeile mit der Versionsnummer des Servers und dem Servernamen ([ServerName](#)) des bedienenden virtuellen Hosts an. Die Einstellung `EMail` erstellt zusätzlich einen "mailto:"-Verweis zum Serveradministrator ([ServerAdmin](#)) des referenzierten Dokuments.

Ab Version 2.0.44 werden die Details der angegebenen Versionsnummer des Servers von der Direktive [ServerTokens](#) kontrolliert.

Siehe auch

- [ServerTokens](#)



Beschreibung:	Konfiguriert den HTTP-Response-Header Server
Syntax:	ServerTokens Major Minor Min[imal] Prod[uctOnly] OS Fu
Voreinstellung:	ServerTokens Full
Kontext:	Serverkonfiguration
Status:	Core
Modul:	core

die Direktive steuert, ob der Response-Header Server, der an den Client zurückgesendet wird, eine Beschreibung des allgemeinen Betriebssystemtyps des Servers wie auch Informationen über einkompilierte Module enthält.

ServerTokens Prod[uctOnly]

Der Server sendet (z.B.): Server : Apache

ServerTokens Major

Der Server sendet (z.B.): Server : Apache/2

ServerTokens Minor

Der Server sendet (z.B.): Server : Apache/2.0

ServerTokens Min[imal]

Der Server sendet (z.B.): Server : Apache/2.0.41

ServerTokens OS

Der Server sendet (z.B.): Server : Apache/2.0.41 (Unix)

ServerTokens Full (oder nicht angegeben)

Der Server sendet (z.B.): Server : Apache/2.0.41 (Unix)
PHP/4.2.2 MyMod/1.2

Diese Einstellung gilt für den gesamten Server und kann nicht auf Virtual-Host-Basis aktiviert oder deaktiviert werden.

Ab Version 2.0.44 steuert diese Direktive auch die Informationen, die

durch die Direktive [ServerSignature](#) angeboten werden.

Siehe auch

- [ServerSignature](#)



Beschreibung:	Erzwingt die Verarbeitung aller passenden Dateien durch einen Handler
Syntax:	SetHandler <i>Handlername</i> None
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	FileInfo
Status:	Core
Modul:	core
Kompatibilität:	Seit Apache 2.0 im Core

Wenn die Direktive innerhalb einer `.htaccess`-Datei oder in einem `<Directory>`- oder `<Location>`-Abschnitt angegeben wird, erzwingt sie, dass alle entsprechenden Dateien von dem durch *Handlername* angegebenen [Handler](#) analysiert werden. Wenn Sie beispielsweise ein Verzeichnis haben, dessen Dateien unabhängig von der Endung gänzlich als Image-Maps interpretiert werden sollen, können Sie folgendes in eine `.htaccess`-Datei in dem Verzeichnis schreiben:

```
SetHandler imap-file
```

Noch ein Beispiel: wenn Sie den Server immer, wenn die URL `http://servername/status` aufgerufen wird, einen Statusbericht anzeigen lassen möchten, dann können Sie folgendes in die `httpd.conf` schreiben:

```
<Location /status>
  SetHandler server-status
</Location>
```

Sie können eine zuvor definierte `SetHandler`-Anweisung aufheben, indem Sie den Wert `None` verwenden.

Siehe auch

- [AddHandler](#)



Beschreibung:	Bestimmt die Filter, die Client-Anfragen und POST-Eingaben verarbeiten
Syntax:	<code>SetInputFilter <i>Filter</i>[;<i>Filter</i>...]</code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
AllowOverride:	FileInfo
Status:	Core
Modul:	core

Die Direktive `SetInputFilter` bestimmt den oder die Filter, die Client-Anfragen und POST-Eingaben verarbeiten, wenn sie vom Server empfangen werden. Diese gelten zusätzlich zu anderweitig definierten Filtern, einschließlich denen der Direktive `AddInputFilter`.

Wenn mehr als ein Filter angegeben wird, dann müssen diese durch Semikolon voneinander getrennt in der Reihenfolge angegeben werden, in der sie die Daten verarbeiten sollen.

Siehe auch

- [Filter](#)-Dokumentation



<u>Beschreibung:</u>	Bestimmt die Filter, die Antworten des Servers verarbeiten
<u>Syntax:</u>	SetOutputFilter <i>Filter</i> [; <i>Filter</i> ...]
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Direktive `SetOutputFilter` bestimmt die Filter, die Antworten des Servers verarbeiten, bevor sie an den Client gesendet werden. Diese gelten zusätzlich zu anderweitig definierten Filtern, einschließlich denen der Direktive `AddOutputFilter`.

Die folgende Konfiguration verarbeitet zum Beispiel alle Dateien im Verzeichnis `/www/data` als Server Side Includes.

```
<Directory /www/data/>
  SetOutputFilter INCLUDES
</Directory>
```

Wenn mehr als ein Filter angegeben wird, dann müssen diese durch Semikolon voneinander getrennt in der Reihenfolge angegeben werden, in der sie die Daten verarbeiten sollen.

Siehe auch

- [Filter](#)-Dokumentation



<u>Beschreibung:</u>	Enable or disable the suEXEC feature
<u>Syntax:</u>	Suexec On Off
<u>Voreinstellung:</u>	On if suexec binary exists with proper owner and mode, Off otherwise
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Available in Apache httpd 2.2.18 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



Beschreibung:	Zeitspanne, die der Server auf verschiedene Ereignisse wartet, bevor er die Anfrage abbricht
Syntax:	Timeout <i>Sekunden</i>
Voreinstellung:	Timeout 300
Kontext:	Serverkonfiguration
Status:	Core
Modul:	core

Die Direktive `Timeout` definiert derzeit die Zeitspanne, die der Apache auf drei Dinge wartet:

1. Die gesamte Zeitspanne, die benötigt wird, um eine GET-Anfrage zu empfangen.
2. Die Zeitspanne zwischen dem Empfang von TCP-Paketen einer POST- oder PUT-Anfrage.
3. Die Zeitspanne zwischen ACKs bei der Übermittlung der TCP-Pakete der Antwort.

Wir haben vor, diese Zeitspannen in Zukunft separat konfigurierbar zu machen. Vor Version 1.2 war der Zeitgeber auf 1200 voreingestellt, wurde dann aber auf 300 herabgesetzt, was immer noch weit mehr ist, als in den meisten Situationen benötigt wird. Die Voreinstellung wurde nicht weiter herabgesetzt, da gelegentlich noch Stellen im Code existieren können, wo der Zeitgeber nicht zurückgesetzt wird, wenn ein Paket verschickt wird.



<u>Beschreibung:</u>	Determines the behaviour on TRACE requests
<u>Syntax:</u>	TraceEnable [<i>on off extended</i>]
<u>Voreinstellung:</u>	TraceEnable on
<u>Kontext:</u>	Serverkonfiguration, Virtual Host
<u>Status:</u>	Core
<u>Modul:</u>	core
<u>Kompatibilität:</u>	Available in Apache 1.3.34, 2.0.55 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



<u>Beschreibung:</u>	Bestimmt, wie der Server seinen eigenen Namen und Port ermittelt
<u>Syntax:</u>	UseCanonicalName On Off DNS
<u>Voreinstellung:</u>	UseCanonicalName Off
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis
<u>Status:</u>	Core
<u>Modul:</u>	core

In vielen Situationen muss der Apache eine *selbstreferenzierende* URL -- d.h. eine URL, die auf den selben Server zurück verweist -- zusammenbauen. Bei `UseCanonicalName On` verwendet der Apache den Hostnamen und Port, der in der `ServerName`-Anweisung angegeben ist, um den kanonischen Namen des Servers zu erstellen. Dieser Name wird in allen selbstreferenzierenden URLs sowie in CGI-Skripten für die Werte von `SERVER_NAME` und `SERVER_PORT` verwendet.

Bei `UseCanonicalName Off` bildet der Apache selbstreferenzierende URLs, indem er den vom Client übermittelten Hostnamen und Port verwendet, sofern diese vorhanden sind (andernfalls wird der kanonische Name, wie oben beschrieben, benutzt). Die Werte sind die gleichen, die zur Anwendung von [namensbasierten virtuellen Hosts](#) verwendet werden, und sie sind mit den gleichen Clients verfügbar (*Anm.d.Ü.:* , die auch in der Lage sind, auf namensbasierte virtuelle Hosts zuzugreifen, d.h. einen Host-Header mitschicken). Die CGI-Variablen `SERVER_NAME` und `SERVER_PORT` werden ebenfalls aus den vom Client angebotenen Werten erstellt.

Ein Intranet-Server, auf den Anwender mit kurzen Namen wie `www` zugreifen, ist ein Beispiel, wo dies sinnvoll sein kann. Sie werden bemerken, dass der Apache den Benutzer auf `http://www.domain.com/sp1at/` umleitet, wenn dieser einen Kurznamen und eine URL, die einem Verzeichnis entspricht, ohne

abschließenden Schrägstrich eingibt, wie z.B. `http://www/splat`. Wenn Sie Authentisierung aktiviert haben, bewirkt dies, dass der Benutzer sich zweimal identifizieren muss (einmal für `www` und noch einmal für `www.domain.com` -- lesen Sie für weitere Informationen [die FAQ zu diesem Thema](#)). Wenn `UseCanonicalName` jedoch auf `Off` gesetzt ist, denn wird der Apache zu `http://www/splat/` umleiten.

Es existiert noch eine dritte Option, `UseCanonicalName DNS`, die für den Betrieb von IP-basierten Massen-Virtual-Hosts gedacht ist, um antiquierte Clients zu unterstützen, die keinen `Host:-Header` bereit stellen. Um selbstreferenzierende URLs zu ermitteln, führt der Apache bei dieser Option ein Reverse-DNS-Lookup auf die IP-Adresse des Servers aus, zu der der Client Verbindung aufgenommen hat.

Warnung

Wenn CGI-Skripte Vermutungen aufgrund des Wertes von `SERVER_NAME` anstellen, können sie durch diese Option fehlschlagen. Clients steht es im Wesentlichen frei, einen Wert für den Hostnamen anzugeben, wie er will. Wenn das CGI-Skript `SERVER_NAME` jedoch lediglich dazu verwendet, selbstreferenzierende URLs zu erstellen, sollte das gerade noch in Ordnung sein.

Siehe auch

- [ServerName](#)
- [Listen](#)



<u>Beschreibung:</u>	Configures how the server determines its own name and port
<u>Syntax:</u>	UseCanonicalPhysicalPort On Off
<u>Voreinstellung:</u>	UseCanonicalPhysicalPort Off
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis
<u>Status:</u>	Core
<u>Modul:</u>	core

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.

Siehe auch

- [UseCanonicalName](#)
- [ServerName](#)
- [Listen](#)



Beschreibung:	Enthält Direktiven, die nur auf bestimmte Hostnamen oder IP-Adressen angewendet werden
Syntax:	<code><VirtualHost Adresse[:Port] [Adresse[:Port]] ...> ... </VirtualHost></code>
Kontext:	Serverkonfiguration
Status:	Core
Modul:	core

`<VirtualHost>` und `</VirtualHost>` werden dazu verwendet, eine Gruppe von Direktiven zusammenzufassen, die nur auf einen bestimmten virtuellen Host angewendet werden. Jede Direktive, die im Virtual-Host-Kontext zulässig ist, kann verwendet werden. Wenn der Server eine Anfrage für ein bestimmtes Dokument eines bestimmten virtuellen Hosts empfängt, dann benutzt er die im `<VirtualHost>`-Container enthaltenen Konfigurationsanweisungen. *Adresse* kann sein:

- Die IP-Adresse des virtuellen Hosts.
- Ein voll qualifizierter Domainname für die IP-Adresse des virtuellen Hosts.
- Das Zeichen `*`, welches nur in Kombination mit `NameVirtualHost *` verwendet wird, um allen IP-Adressen zu entsprechen.
- Die Zeichenkette `_default_`, die nur mit IP-basierten virtuellen Hosts verwendet wird, um nicht zugewiesene IP-Adressen aufzufangen.

Beispiel

```
<VirtualHost 10.1.2.3>
  ServerAdmin webmaster@host.foo.com
  DocumentRoot /www/docs/host.foo.com
  ServerName host.foo.com
  ErrorLog logs/host.foo.com-error_log
  TransferLog logs/host.foo.com-access_log
</VirtualHost>
```

IPv6-Adressen müssen in eckigen Klammern angegeben werden, da die optionale Portnummer sonst nicht erkannt werden kann. Hier ein IPv6-Beispiel:

```
<VirtualHost [2001:db8::a00:20ff:fea7:ccea]>  
  ServerAdmin webmaster@host.example.com  
  DocumentRoot /www/docs/host.example.com  
  ServerName host.example.com  
  ErrorLog logs/host.example.com-error_log  
  TransferLog logs/host.example.com-access_log  
</VirtualHost>
```

Jeder virtuelle Host muss einer anderen IP-Adresse, einem anderen Port oder einem anderen Hostnamen für den Server entsprechen. Im ersten Fall muss die Servermaschine so eingerichtet sein, dass sie IP-Pakete für mehrere Adressen akzeptiert. (Wenn der Rechner nicht mehrere Netzwerkkarten besitzt, kann dies mit dem Befehl `ifconfig alias` durchgeführt werden -- sofern Ihr Betriebssystem das unterstützt).

Anmerkung

Die Verwendung von `<VirtualHost>` beeinflusst **nicht**, an welchen Adressen der Apache lauscht. Sie müssen mit `Listen` sicherstellen, dass der Apache an der richtigen Adresse lauscht.

Bei der Verwendung IP-basierter virtueller Hosts kann der spezielle Name `_default_` benutzt werden. In diesem Fall weist der Apache jede IP-Adresse diesem virtuellen Host zu, die nicht explizit in einem anderen virtuellen Host angegeben ist. Falls kein virtueller Host `_default_` angegeben ist, wird die "Hauptserver"-Konfiguration, die aus allen Definitionen außerhalb der Virtual-Host-Abschnitte besteht, für nicht passende IPs verwendet. (Beachten Sie jedoch, dass eine IP-Adressen die zu einer `NameVirtualHost`-Anweisung passt, weder den "Hauptserver" noch den virtuellen Host `_default_` verwendet. Lesen Sie für weitere Details die Dokumentation zu [namensbasierten virtuell Hosts](#).)

Sie können einen speziellen `:Port` angeben, um den entsprechenden Port zu wechseln. Falls nicht angegeben, wird er auf den gleichen Port voreingestellt, wie die letzte [Listen](#)-Anweisung des Hauptservers. Sie können auch `:*` angeben, um alle Ports dieser Adresse zu akzeptieren. (Dies wird zusammen mit `_default_` empfohlen.)

Sicherheit

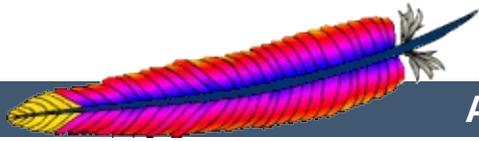
Lesen Sie das Dokument [Sicherheitshinweise](#) für Details, warum Ihre Sicherheit gefährdet sein kann, wenn das Verzeichnis, in dem Protokolldateien gespeichert werden, für jemanden anderes als den Benutzer beschreibbar ist, der den Server gestartet hat.

Siehe auch

- [Apache-Dokumentation zu virtuellen Hosts](#)
- [Probleme bezüglich DNS und Apache](#)
- [Bestimmen, welche Adressen und Ports der Apache verwendet](#)
- [Wie die Abschnitte `<Directory>`, `<Location>` und `<Files>` arbeiten](#) für eine Erläuterung, wie diese verschiedenen Abschnitte miteinander kombiniert werden, wenn eine Anfrage empfangen wird

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Allgemeine Direktiven der Apache-MPMs

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Beschreibung: Eine Sammlung von Direktiven, die in mehr als einem Multi-Processing-Modul (MPM) implementiert sind.

Status: MPM



<u>Beschreibung:</u>	Vom Apache verwendete Methode zur Serialisierung mehrerer Kindprozesse, die Anfragen an Netzwerk-Sockets entgegennehmen.
<u>Syntax:</u>	AcceptMutex Default <i>Methode</i>
<u>Voreinstellung:</u>	AcceptMutex Default
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>leader</u> , <u>perchild</u> , <u>prefork</u> , <u>threadpool</u> , <u>worker</u>

Die Direktive `AcceptMutex` bestimmt die Methode, die der Apache zur Serialisierung mehrerer Kindprozesse verwendet, welche Anfragen an Netzwerk-Sockets entgegennehmen. Vor Apache 2.0 war diese Methode nur zur Kompilierungszeit einstellbar. Die optimale Methode ist sehr stark von der Architektur und Plattform abhängig. Lesen Sie bitte [Performance-Hinweise](#) für weitere Details.

Wenn die Direktive auf `Default` eingestellt ist, dann wird die zur Kompilierungszeit gewählte Voreinstellung verwendet. Weitere mögliche Methoden sind unten angegeben. Beachten Sie, dass nicht alle Methoden auf allen Plattformen verfügbar sind. Wird eine Methode angegeben, die nicht verfügbar ist, dann wird eine Nachricht in das Fehlerprotokoll geschrieben, welche die verfügbaren Methoden auflistet.

flock

verwendet die Systemfunktion `flock(2)`, um die durch die [LockFile](#)-Direktive definierte Datei zu sperren.

fcntl

verwendet die Systemfunktion `fcntl(2)`, um die durch die [LockFile](#)-Direktive definierte Datei zu sperren.

posixsem

verwendet POSIX-kompatible Semaphore, um den Mutex zu

implementieren.

pthread

verwendet gemäß der POSIX-Thread-Spezifikation implementierte POSIX-Mutexe.

sysvsem

verwendet Semaphoren des SysV-Typs, um den Mutex zu implementieren.

Um die bei der Kompilierung gewählte Voreinstellung für Ihr System herauszufinden, können Sie Ihr [LogLevel](#) auf debug setzen. Dann wird der voreingestellte [AcceptMutex](#) ins [ErrorLog](#) geschrieben.

Warnung

Auf den meisten Systemen stoppt der Server mit der Beantwortung von Anfragen, wenn die Option `pthread` ausgewählt wurde und ein Kindprozess unkontrolliert endet während er den `AcceptCntl`-Mutex hält. In diesem Fall muss der Server manuell neu gestartet werden, um wieder weiter zu arbeiten.

Eine Ausnahme stellt Solaris dar, da es einen Mechanismus anbietet, den der Apache verwendet und der üblicherweise die Freigabe des Mutex erlaubt, nachdem ein Kindprozess, der gerade einen Mutex hält, abgestürzt ist.

Wenn Ihr System die Funktion `pthread_mutexattr_setrobust_np()` bereitstellt, können Sie wahrscheinlich die Option `pthread` problemlos verwenden.



<u>Beschreibung:</u>	Directory for apache to run chroot(8) after startup.
<u>Syntax:</u>	ChrootDir <i>/path/to/directory</i>
<u>Voreinstellung:</u>	none
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	event , prefork , worker
<u>Kompatibilität:</u>	Available in Apache 2.2.10 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



<u>Beschreibung:</u>	Verzeichnis, in das der Apache zu wechseln versucht, bevor er einen Hauptspeicherauszug erstellt
<u>Syntax:</u>	<code>CoreDumpDirectory Verzeichnis</code>
<u>Voreinstellung:</u>	Für die Voreinstellung siehe Beschreibung
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	beos , leader , mpm_winnt , perchild , prefork , threadpool , worker

Dies beeinflusst das Verzeichnis, in welches der Apache zu wechseln versucht, bevor er einen Hauptspeicherauszug (*Anm.d.Ü.:* einen so genannten Core-Dump) erstellt. Die Voreinstellung ist das **ServerRoot**-Verzeichnis. Da dieses jedoch nicht für den Benutzer beschreibbar sein soll, unter dem der Server läuft, werden normalerweise keine Hauptspeicherauszüge geschrieben. Wenn Sie zum Debuggen einen Hauptspeicherauszug haben möchten, können Sie ihn mit dieser Direktive an einem anderen Ort ablegen lassen.

Hauptspeicherauszüge unter Linux

Wenn Apache als root startet und zu einem anderen Benutzer wechselt, *deaktiviert* der Linux-Kernel Hauptspeicherauszüge auch dann, wenn der Prozess in dem Verzeichnis schreiben darf. Ab Linux 2.4 reaktiviert Apache (ab 2.0.46) Hauptspeicherauszüge wieder, jedoch nur dann, wenn Sie explizit **CoreDumpDirectory** konfigurieren.



<u>Beschreibung:</u>	Aktiviert einen Hook, der nach einem Absturz noch Ausnahmefehler behandeln lassen kann
<u>Syntax:</u>	EnableExceptionHook On Off
<u>Voreinstellung:</u>	EnableExceptionHook Off
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	leader , perchild , prefork , threadpool , worker
<u>Kompatibilität:</u>	Verfügbar seit Version 2.0.49

Diese Direktive ist aus Sicherheitsgründen nur verfügbar, wenn der Server mit der Option `--enable-exception-hook` konfiguriert wurde. Sie aktiviert einen Hook, der es externen Modulen erlaubt, sich dort einzuhängen und nach dem Absturz eines Kindprozesses noch Aktionen durchzuführen.

Es existieren bereits zwei Module, `mod_whatkilledus` und `mod_backtrace`, welche diesen Hook verwenden. Weitere Informationen hierzu finden Sie auf Jeff Trawicks [EnableExceptionHook-Seite](#).



<u>Beschreibung:</u>	Specify a timeout after which a gracefully shutdown server will exit.
<u>Syntax:</u>	<code>GracefulShutdownTimeout</code> <i>seconds</i>
<u>Voreinstellung:</u>	<code>GracefulShutdownTimeout</code> 0
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	prefork , worker , event
<u>Kompatibilität:</u>	Available in version 2.2 and later

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



<u>Beschreibung:</u>	Benutzergruppe, unter welcher der Server Anfragen beantwortet
<u>Syntax:</u>	Group <i>Unix-Gruppe</i>
<u>Voreinstellung:</u>	Group #-1
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>beos</u> , <u>leader</u> , <u>mpmt_os2</u> , <u>perchild</u> , <u>prefork</u> , <u>threadpool</u> , <u>worker</u>
<u>Kompatibilität:</u>	Seit Apache 2.0 nur in der globalen Server-Konfiguration gültig

Die Direktive **Group** bestimmt die Benutzergruppe, unter welcher der Server Anfragen beantwortet. Um diese Direktive zu verwenden, muss der Server als root gestartet werden. Wenn Sie den Server unter einem nicht-root-Benutzer starten, wird er nicht zur angegebenen Gruppe wechseln können und statt dessen weiter mit der Gruppe des ursprünglichen Benutzers laufen. *Unix-Gruppe* kann sein:

Ein Gruppenname

Verweist auf die durch den Namen angegebene Gruppe.

gefolgt von einer Gruppennummer.

Verweist auf die durch ihre Nummer angegebene Gruppe.

Beispiel

```
Group www-group
```

Es wird empfohlen, dass Sie eine neue Gruppe speziell zum Betrieb des Servers erstellen. Einige Administratoren verwenden den Benutzer nobody. Dies ist jedoch nicht immer möglich oder gewünscht.

Sicherheit

Setzen Sie **Group** (oder **User**) nicht auf `root`, solange Sie nicht ganz genau wissen, was Sie tun und welche Gefahren Sie eingehen.

Wichtiger Hinweis: Die Verwendung der Direktive innerhalb von `<VirtualHost>` wird nicht länger unterstützt. Benutzen Sie **SuexecUserGroup** um Ihren Server für `suexec` einzurichten.

Anmerkung

Obwohl die Direktive **Group** in den MPMs `beos` und `mpmt_os2` existiert, ist sie dort tatsächlich eine Leeranweisung und existiert nur aus Kompatibilitätsgründen.



Beschreibung:	IP-Adressen und Ports, an denen der Server lauscht
Syntax:	Listen [<i>IP-Adresse:</i>]Port
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	beos , leader , mpm_netware , mpm_winnt , mpmt_os2 , perchild , prefork , threadpool , worker
Kompatibilität:	Seit Apache 2.0 vorgeschrieben

Die Direktive **Listen** weist den Apache an, nur an den angegebenen IP-Adressen oder Ports zu lauschen. Standardmäßig antwortet er auf alle Anfragen an allen IP-Interfaces. **Listen** ist nun eine notwendige Anweisung. Wenn sie nicht in der Konfigurationsdatei enthalten ist, wird der Server-Start fehlschlagen. Dies ist eine Änderung gegenüber früheren Versionen des Apache.

Die Direktive **Listen** weist den Server an, ankommende Anfragen am angegebenen Port oder der Kombination aus Adresse und Port entgegenzunehmen. Wenn nur eine Portnummer angegeben ist, dann lauscht der Server am angegebenen Port an allen Interfaces. Wenn sowohl eine IP-Adresse als auch ein Port angegeben sind, dann lauscht der Server am angegebenen Port und Interface.

Es können mehrere **Listen**-Anweisungen verwendet werden, um eine Reihe von Adressen und Port anzugeben, an denen gelauscht werden soll. Der Server antwortet auf Anfragen von jedem der aufgeführten Adressen und Ports.

Um beispielsweise den Server Verbindungen an den beiden Ports 80 und 8000 annehmen zu lassen, verwenden Sie:

```
Listen 80
Listen 8000
```

Um den Server Verbindungen an zwei angegebenen Interfaces und Ports annehmen zu lassen, verwenden Sie:

```
Listen 192.170.2.1:80  
Listen 192.170.2.5:8000
```

IPv6-Adressen müssen wie in dem folgenden Beispiel in eckige Klammern eingeschlossen werden:

```
Listen [2001:db8::a00:20ff:fea7:ccea]:80
```

Fehlermöglichkeit

Mehrere `Listen`-Direktiven für gleiche IP-Adresse und Port führen zur Fehlermeldung `Address already in use` (*Anm.d.Ü.:* Adresse schon in Benutzung).

Siehe auch

- [DNS-Probleme](#)
- [Bestimmen, welche Adressen und Ports der Apache verwendet](#)



<u>Beschreibung:</u>	Maximale Länge der Warteschlange schwebender Verbindungen
<u>Syntax:</u>	ListenBacklog <i>backlog</i>
<u>Voreinstellung:</u>	ListenBacklog 511
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>beos</u> , <u>leader</u> , <u>mpm_network</u> , <u>mpm_winnt</u> , <u>mpmt_os2</u> , <u>perchild</u> , <u>prefork</u> , <u>threadpool</u> , <u>worker</u>

Die maximale Länge der Warteschlange schwebender Verbindungen. Üblicherweise ist keine Feineinstellung notwendig oder sinnvoll, auf einigen System kann es jedoch gewünscht sein, diesen Wert bei TCP-SYN-Angriffen zu erhöhen. Beachten Sie auch die Beschreibung des backlog-Parameters der Systemfunktion `listen(2)`.

Der Wert wird vom Betriebssystem oft auf eine niedrigere Einstellung begrenzt. Dies variiert von Betriebssystem zu Betriebssystem. Beachten Sie auch, dass viele Betriebssysteme nicht genau beachten, was für backlog angegeben ist, jedoch einen Wert basierend auf der Angabe (normalerweise jedoch größer als diese) verwenden.



Beschreibung:	Ablageort der Lock-Datei für die Serialisierung von entgegengenommenen Anfragen
Syntax:	LockFile <i>Dateiname</i>
Voreinstellung:	LockFile logs/accept.lock
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	leader , perchild , prefork , threadpool , worker

Die Direktive `LockFile` legt den Pfad zur Lock-Datei fest, die verwendet wird, wenn der Apache mit einer der `AcceptMutex`-Einstellungen `fcntl` oder `flock` verwendet wird. Die Anweisung sollte normalerweise bei der Voreinstellung belassen werden. Der Hauptgrund, sie zu ändern, ist, wenn das `logs`-Verzeichnis auf einem per NFS-eingebundenen Laufwerk liegt, da **die Lock-Datei auf einer lokalen Platte abgelegt sein muss**. Die PID (*Anm.d.Ü.:* Prozess-ID) des Hauptserverprozesses wird automatisch an den Dateinamen angehängt.

Sicherheit

Es ist am besten, die Ablage in einem allgemein (*Anm.d.Ü.:* für jedermann) beschreibbaren Verzeichnis wie `/var/tmp` zu vermeiden, da ein Denial-of-Service-Angriff gestartet werden könnte und der Server am Start gehindert werden könnte, indem eine Lock-Datei mit dem gleichen Namen erstellt wird, wie der Server sie zu erstellen versuchen würde.

Siehe auch

- [AcceptMutex](#)



<u>Beschreibung:</u>	Maximale Anzahl der Kindprozesse, die zur Bedienung von Anfragen gestartet wird
<u>Syntax:</u>	<code>MaxClients <i>Anzahl</i></code>
<u>Voreinstellung:</u>	Für Details siehe Beschreibung
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<code><u>beos</u></code> , <code><u>leader</u></code> , <code><u>prefork</u></code> , <code><u>threadpool</u></code> , <code><u>worker</u></code>

Die Direktive `MaxClients` setzt die Grenze für die Anzahl gleichzeitig bedienter Anfragen. Jeder Verbindungsversuch oberhalb der `MaxClients`-Begrenzung wird üblicherweise in eine Warteschlange gestellt, bis zu einer Anzahl basierend auf der `ListenBacklog`-Anweisung. Sobald ein Kindprozess am Ende einer anderen Anfrage freigegeben wird, wird die Verbindung bedient.

Für Server ohne Thread-Unterstützung (z.B. `prefork`) wird `MaxClients` als maximale Anzahl der Kindprozesse verstanden, die zur Bedienung von Anfragen gestartet werden. Die Voreinstellung ist 256. Um diesen Wert zu erhöhen, muss auch `ServerLimit` angehoben werden.

Bei Servern mit Thread-Unterstützung und bei Hybrid-Servern (z.B. `beos` oder `worker`) begrenzt `MaxClients` die Gesamtzahl der Threads, die für die Bedienung von Anfragen verfügbar sind. Die Voreinstellung für `beos` ist 50. Bei Hybrid-MPMs ist die Voreinstellung 16 (`ServerLimit`) multipliziert mit dem Wert 25 (`ThreadsPerChild`). Um `MaxClients` auf einen Wert zu erhöhen, der mehr als 16 Prozesse erfordert, müssen Sie daher auch `ServerLimit` anheben.



<u>Beschreibung:</u>	Maximale Menge des Arbeitsspeichers, den die Haupt-Zuteilungsroutine verwalten darf, ohne <code>free()</code> aufzurufen
<u>Syntax:</u>	<code>MaxMemFree</code> <i>KBytes</i>
<u>Voreinstellung:</u>	<code>MaxMemFree</code> 0
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u><code>beos</code></u> , <u><code>leader</code></u> , <u><code>mpm_network</code></u> , <u><code>prefork</code></u> , <u><code>threadpool</code></u> , <u><code>worker</code></u> , <u><code>mpm_winnt</code></u>

Die Direktive `MaxMemFree` gibt die maximale Menge freier Kilobytes an, welche die Haupt-Zuteilungsroutine verwalten darf, ohne `free()` aufzurufen. Wenn keine Angabe gemacht wird, oder Null angegeben ist, wird dieser Wert nicht eingeschränkt.



Beschreibung:	Obergrenze für die Anzahl von Anfragen, die ein einzelner Kindprozess während seines Lebens bearbeitet
Syntax:	<code>MaxRequestsPerChild number</code>
Voreinstellung:	<code>MaxRequestsPerChild 10000</code>
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	<code>leader</code> , <code>mpm_netware</code> , <code>mpm_winnt</code> , <code>mpmt_os2</code> , <code>perchild</code> , <code>prefork</code> , <code>threadpool</code> , <code>worker</code>

Die Direktive `MaxRequestsPerChild` legt die Grenze für die Anzahl von Anfragen fest, die ein einzelner Kinprozess während seines Lebens bearbeitet. Nach `MaxRequestsPerChild` Anfragen stirbt der Kindprozess. Wenn `MaxRequestsPerChild 0` ist, endet der Prozess niemals.

Abweichende Voreinstellungen

Die Voreinstellung für `mpm_netware` und `mpm_winnt` ist 0.

Die Begrenzung von `MaxRequestsPerChild` auf einen Wert ungleich Null hat zwei vorteilhafte Auswirkungen:

- sie begrenzt die Menge an Arbeitsspeicher, die ein Prozess durch (versehentliche) Speicherlecks verbrauchen kann.
- das Festlegen einer endlichen Lebensdauer von Prozessen hilft, die Anzahl von Prozessen zu reduzieren, wenn die Serverlast zurückgeht.

Anmerkung

Bei `KeepAlive`-Anfragen wird nur die erste Anfrage für diese begrenzung gezählt. Eigentlich wird nur die Begrenzung für die Anzahl

der *Verbindungen* pro Kindprozess geändert.



Beschreibung:	Maximale Anzahl unbeschäftigter Threads
Syntax:	MaxSpareThreads <i>Anzahl</i>
Voreinstellung:	Für Details siehe Beschreibung
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	beos , leader , mpm_network , mpm_os2 , perchild , threadpool , worker

Maximale Anzahl unbeschäftigter Threads. Die verschiedenen MPMs behandeln diese Anweisung unterschiedlich.

Die Voreinstellung für [perchild](#) ist `MaxSpareThreads 10`. Das MPM überwacht die Anzahl der unbeschäftigten Threads auf der Basis einzelner Kindprozesse. Wenn zu viele unbeschäftigte Threads in einem Kindprozess existieren, beendet der Server Threads innerhalb dieses Kindprozesses.

Die Voreinstellung für [worker](#), [leader](#) und [threadpool](#) ist `MaxSpareThreads 250`. Diese MPMs behandeln Threads auf einer serverweiten Basis. Wenn zu viele unbeschäftigte Threads im Server existieren, dann werden solange Kindprozesse beendet, bis die Anzahl der unbeschäftigten Threads kleiner als der angegebene Wert ist.

Die Voreinstellung für [mpm_network](#) ist `MaxSpareThreads 100`. Da dieses MPM nur einen einzigen Prozess ausführt, ist die Zählung überschüssiger Threads ebenfalls serverweit.

[beos](#) and [mpm_os2](#) arbeiten ähnlich wie [mpm_network](#). Die Voreinstellung für [beos](#) ist `MaxSpareThreads 50`. Die Voreinstellung für [mpm_os2](#) ist 10.

Restriktionen

Der Wertebereich von `MaxSpareThreads` ist eingeschränkt. Apache korrigiert den angegebenen Wert automatisch gemäß den folgenden Regeln:

- `perchild` verlangt, dass `MaxSpareThreads` kleiner oder gleich `ThreadLimit` ist.
- `mpm_netware` verlangt einen Wert größer als `MinSpareThreads`.
- Bei `leader`, `threadpool` und `worker` muss der Wert größer oder gleich der Summe aus `MinSpareThreads` und `ThreadsPerChild` sein.

Siehe auch

- [MinSpareThreads](#)
- [StartServers](#)



Beschreibung:	Minimale Anzahl unbeschäftigter Threads, die zur Bedienung von Anfragespitzen zur Verfügung stehen
Syntax:	MinSpareThreads <i>Anzahl</i>
Voreinstellung:	Für Details siehe Beschreibung
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	beos , leader , mpm_netware , mpmt_os2 , perchild , threadpool , worker

Minimale Anzahl unbeschäftigter Threads, um Anfragespitzen zu bedienen. Die verschiedenen MPMs behandeln die Anweisung unterschiedlich.

[perchild](#) verwendet die Voreinstellung `MinSpareThreads 5` und überwacht die Anzahl der unbeschäftigten Threads auf der Basis einzelner Kindprozesse. Wenn in einem Kindprozess nicht genügend unbeschäftigte Threads vorhanden sind, erstellt der Server neue Threads innerhalb dieses Kindprozesses. Wenn Sie also [NumServers](#) auf 10 und [MinSpareThreads](#) auf einen Wert von 5 setzen, haben Sie mindestens 50 unbeschäftigte Threads auf Ihrem System.

[worker](#), [leader](#) und [threadpool](#) verwenden eine Voreinstellung von `MinSpareThreads 75` und behandeln unbeschäftigte Threads auf serverweiter Basis. Wenn nicht genügend unbeschäftigte Threads im Server vorhanden sind, dann werden solange Kindprozesse erzeugt, bis die Anzahl unbeschäftigter Threads größer als der angegebene Wert ist.

[mpm_netware](#) verwendet die Voreinstellung `MinSpareThreads 10` und verfolgt dies serverweit, da es ein Einzelprozess-MPM ist.

[beos](#) und [mpmt_os2](#) arbeiten ähnlich wie [mpm_netware](#). Die Voreinstellung für [beos](#) ist `MinSpareThreads 1`. Die Voreinstellung für [mpmt_os2](#) ist 5.

Siehe auch

- [MaxSpareThreads](#)
- [StartServers](#)



Beschreibung:	Datei, in welcher der Server die Prozess-ID des Daemons ablegt
Syntax:	<code>PidFile <i>Dateiname</i></code>
Voreinstellung:	<code>PidFile logs/httpd.pid</code>
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	beos , leader , mpm_winnt , mpmt_os2 , perchild , prefork , threadpool , worker

Die Direktive `PidFile` bestimmt die Datei, in welcher der Server die Prozess-ID des Daemons ablegt. Wenn der Dateiname nicht absolut angegeben wird, wird er relativ zu `ServerRoot` interpretiert.

Beispiel

```
PidFile /var/run/apache.pid
```

Es ist oft hilfreich, dem Server ein Signal senden zu können, damit er seine `ErrorLogs` und `TransferLogs` schließt und dann neu öffnet und seine Konfigurationsdateien neu einliest. Dies kann durch Senden eines SIGHUP-Signals (kill -1) an die Prozess-ID geschehen, die im `PidFile` eingetragen ist.

Die `PidFile`-Datei unterliegt den gleichen Warnungen über die Ablage von Protokolldateien und [Sicherheit](#).

Anmerkung

Ab Apache 2 wird empfohlen, nur das Skript `apachectl` zum (Neu-)Starten und Stoppen des Servers zu verwenden.



<u>Beschreibung:</u>	TCP receive buffer size
<u>Syntax:</u>	ReceiveBufferSize <i>bytes</i>
<u>Voreinstellung:</u>	ReceiveBufferSize 0
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	beos , mpm_netware , mpm_winnt , mpmt_os2 , prefork , worker

Die Dokumentation zu dieser Direktive wurde noch nicht übersetzt. Bitte schauen Sie in die englische Version.



<u>Beschreibung:</u>	Ablageort der Datei, die zur Speicherung von Daten zur Koordinierung der Kindprozesse verwendet wird
<u>Syntax:</u>	ScoreBoardFile <i>Dateipfad</i>
<u>Voreinstellung:</u>	ScoreBoardFile logs/apache_status
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	beos , leader , mpm_winnt , perchild , prefork , threadpool , worker

Apache verwendet ein Scoreboard zur Kommunikation zwischen seinen Eltern- und Kindprozessen. Einige Architekturen erfordern eine Datei zur Unterstützung der Kommunikation. Wenn die Datei undefiniert bleibt, versucht der Apache zuerst, das Scoreboard im Arbeitsspeicher zu erstellen (Verwendung von anonymem Shared-Memory), und versucht bei einem Fehlschlag anschließend die Datei auf der Festplatte zu erstellen (Verwendung von Datei-basiertem Shared-Memory). Die Angabe dieser Direktive veranlaßt den Apache stets, die Datei auf der Festplatte zu erstellen.

Beispiel

```
ScoreBoardFile /var/run/apache_status
```

Datei-basiertes Shared-Memory ist für Applikationen von Drittanbietern hilfreich, die direkten Zugriff auf das Scoreboard benötigen.

Wenn Sie eine **ScoreBoardFile**-Anweisung verwenden, erreichen Sie eventuell eine höhere Geschwindigkeit, wenn Sie die Datei auf einer RAM-Disk ablegen. Achten Sie darauf, die gleichen Warnungen wie über die Ablage von Protokolldateien und [Sicherheit](#) zu beherzigen.

Siehe auch

- [Apache beenden und neu starten](#)



<u>Beschreibung:</u>	Größe des TCP-Puffers
<u>Syntax:</u>	SendBufferSize <i>Bytes</i>
<u>Voreinstellung:</u>	SendBufferSize 0
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>beos</u> , <u>leader</u> , <u>mpm_netware</u> , <u>mpm_winnt</u> , <u>mpmt_os2</u> , <u>perchild</u> , <u>prefork</u> , <u>threadpool</u> , <u>worker</u>

Der Server setzt die Größe des TCP-Puffers auf die angegebene Anzahl Bytes. Dies ist sehr hilfreich, um Voreinstellungen alter Standardbetriebssysteme für Hochgeschwindigkeitsverbindungen mit hoher Latenzzeit anzuheben (*d.h.* 100ms oder so, wie bei Interkontinentalverbindungen).

Wird der Wert auf 0 gesetzt, dann verwendet der Server die Voreinstellung des Betriebssystems.



<u>Beschreibung:</u>	Obergrenze für die konfigurierbare Anzahl von Prozessen
<u>Syntax:</u>	<code>ServerLimit <i>Anzahl</i></code>
<u>Voreinstellung:</u>	Für Details siehe Beschreibung
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>leader</u> , <u>perchild</u> , <u>prefork</u> , <u>threadpool</u> , <u>worker</u>

Bei dem MPM [prefork](#) bestimmt die Direktive den während der Lebensdauer des Apache-Prozesses maximal einstellbaren Wert für [MaxClients](#). Beim MPM [worker](#) bestimmt die Direktive in Verbindung mit [ThreadLimit](#) den Maximalwert für [MaxClients](#) für die Lebensdauer des Apache-Prozesses. Jeder Versuch, diese Anweisung während eines Neustarts zu ändern, wird ignoriert. [MaxClients](#) kann jedoch während eines Neustarts geändert werden.

Lassen Sie besondere Vorsicht bei der Verwendung dieser Direktive walten. Wenn [ServerLimit](#) auf einen Wert deutlich höher als notwendig gesetzt wird, wird zusätzliches, unbenutztes Shared-Memory belegt. Wenn sowohl [ServerLimit](#) als auch [MaxClients](#) auf Werte gesetzt werden, die größer sind, als das System sie handhaben kann, dann kann der Apache möglicherweise nicht starten, oder das System kann instabil werden.

Verwenden Sie die Direktive bei dem MPM [prefork](#) nur, wenn Sie [MaxClients](#) auf mehr als 256 (Voreinstellung) setzen müssen. Setzen Sie den Wert nicht höher als den Wert, den Sie für [MaxClients](#) angeben möchten.

Verwenden Sie die Direktive bei [worker](#), [leader](#) und [threadpool](#) nur, wenn Ihre [MaxClients](#)- und [ThreadsPerChild](#)-Einstellungen mehr

als 16 Serverprozesse (Voreinstellung) erfordern. Setzen Sie den Wert dieser Direktive nicht höher, als die Anzahl der Serverprozesse, die dafür erforderlich ist, was Sie bei [MaxClients](#) und [ThreadsPerChild](#) angeben möchten.

Verwenden Sie die Direktive beim MPM [perchild](#) nur, wenn Sie [NumServers](#) auf einen Wert größer als 8 (Voreinstellung) setzen müssen.

Anmerkung

Eine feste Begrenzung von `ServerLimit 20000` ist in den Server einkompiliert (bei dem MPM [prefork](#) 200000). Dies soll unangenehme Effekte durch Tippfehler verhindern.

Siehe auch

- [Apache beenden und neu starten](#)



<u>Beschreibung:</u>	Anzahl der Kindprozesse des Servers, die beim Start erstellt werden
<u>Syntax:</u>	StartServers <i>Anzahl</i>
<u>Voreinstellung:</u>	Für Details siehe Beschreibung
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>leader</u> , <u>mpmt_os2</u> , <u>prefork</u> , <u>threadpool</u> , <u>worker</u>

Die Direktive **StartServers** bestimmt die Anzahl der Kindprozesse des Servers, die beim Start erstellt werden. Da die Anzahl der Prozesse abhängig von der Last dynamisch kontrolliert wird, besteht normalerweise wenig Grund für eine Änderung dieses Parameters.

Die Voreinstellung unterscheidet sich von MPM zu MPM. Bei leader, threadpool und worker ist die Voreinstellung StartServers 3. Die Voreinstellung bei prefork ist 5 und bei mpmt_os2 2.



Beschreibung:	Anzahl der Threads, die beim Start erstellt werden
Syntax:	<code>StartThreads</code> <i>Anzahl</i>
Voreinstellung:	Für Details siehe Beschreibung
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	beos , mpm_netware , perchild

Anzahl der Threads, die beim Start erstellt werden. Da die Anzahl der Threads abhängig von der Last dynamisch kontrolliert wird, besteht normalerweise wenig Grund für eine Änderung dieses Parameters.

Die Voreinstellung für [perchild](#) ist `StartThreads 5`. Die Direktive setzt während des Starts die Anzahl der Threads pro Prozess.

Die Voreinstellung bei [mpm_netware](#) ist `StartThreads 50`. Da hier lediglich ein einzelner Prozess existiert, ist dies die Gesamtzahl der Threads, die beim Start erstellt wird, um Anfragen zu bedienen.

Die Voreinstellung für [beos](#) ist `StartThreads 10`. Die Einstellung reflektiert ebenfalls die Gesamtzahl der Threads, die beim Start erstellt werden, um Anfragen zu bedienen.



<u>Beschreibung:</u>	Bestimmt die Obergrenze der konfigurierbaren Anzahl von Threads pro Kindprozess
<u>Syntax:</u>	ThreadLimit <i>Anzahl</i>
<u>Voreinstellung:</u>	Für Details siehe Beschreibung
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	leader , mpm_winnt , perchild , threadpool , worker
<u>Kompatibilität:</u>	Verfügbar für mpm_winnt ab Apache 2.0.41

Die Direktive bestimmt den während der Lebensdauer des Apache-Prozesses maximal einstellbaren Wert für [ThreadsPerChild](#). Jeder Versuch, diese Direktive während eines Neustarts zu ändern, wird ignoriert. [ThreadsPerChild](#) kann jedoch während eines Neustarts modifiziert werden bis zu dem Wert dieser Anweisung.

Lassen Sie besondere Vorsicht bei der Verwendung dieser Direktive walten. Wenn [ThreadLimit](#) auf einen Wert deutlich höher als [ThreadsPerChild](#) gesetzt wird, wird zusätzliches, ungenutztes Shared-Memory belegt. Wenn sowohl [ThreadLimit](#) als auch [ThreadsPerChild](#) auf Werte gesetzt werden, die größer sind, als das System sie handhaben kann, dann kann der Apache möglicherweise nicht starten oder das System kann instabil werden. Setzen Sie den Wert dieser Direktive nicht höher als Ihre größte erwartete Einstellung für [ThreadsPerChild](#) während der aktuellen Ausführung des Apache.

Die Voreinstellung für [ThreadLimit](#) ist 1920 wenn sie zusammen mit [mpm_winnt](#) verwendet wird, und 64 bei der Verwendung mit anderen MPMs.

Anmerkung

Eine feste Begrenzung von ThreadLimit 20000 (oder ThreadLimit 15000 bei mpm_winnt) ist in den Server einkompiliert. Dies soll unangenehme Effekte durch Tippfehler verhindern.



<u>Beschreibung:</u>	Anzahl der Threads, die mit jedem Kindprozess gestartet werden
<u>Syntax:</u>	ThreadsPerChild <i>Anzahl</i>
<u>Voreinstellung:</u>	Für Details siehe Beschreibung
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>leader</u> , <u>mpm_winnt</u> , <u>threadpool</u> , <u>worker</u>

Die Direktive legt die Anzahl der Threads fest, die mit jedem Kindprozess gestartet werden. Der Kindprozess erstellt diese Threads beim Start und erstellt später keine weiteren mehr. Wenn Sie ein MPM wie mpm_winnt verwenden, wo nur ein Kindprozess existiert, dann sollte diese Angabe hoch genug sein, die gesamte Last des Servers zu bewältigen. Wenn Sie ein MPM wie worker verwenden, wo mehrere Kindprozesse existieren, dann sollte die *Gesamtzahl* der Thread groß genug sein, die übliche Last auf dem Server zu bewältigen.

Die Voreinstellung für `ThreadsPerChild` ist 64, wenn mpm_winnt verwendet wird, und 25 bei der Verwendung der anderen MPMs.



<u>Beschreibung:</u>	Die Größe des Stacks in Bytes, der von Threads verwendet wird, die Client-Verbindungen bearbeiten.
<u>Syntax:</u>	<code>ThreadStackSize size</code>
<u>Voreinstellung:</u>	65536 unter NetWare; variiert bei anderen Betriebssystemen.
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<u>leader</u> , <u>mpm_netware</u> , <u>mpm_winnt</u> , <u>perchild</u> , <u>threadpool</u> , <u>worker</u>
<u>Kompatibilität:</u>	Verfügbar seit Version 2.1

Die Direktive `ThreadStackSize` legt die Größe des Stacks (für Autodaten) der Threads fest, die Client-Verbindungen bearbeiten und Module aufrufen, welche bei der Verarbeitung dieser Verbindungen helfen. In den meisten Fällen ist die Voreinstellung des Betriebssystems angemessen, doch unter bestimmten Umständen kann es sinnvoll sein, den Wert anzupassen:

- Auf Plattformen mit einer relativ kleinen Voreingestellung für die Größe des Thread-Stacks (z.B. HP-UX) kann der Apache bei der Verwendung einiger Drittanbietermodule, die einen relativ hohen Bedarf an Speicherplatz haben, abstürzen. Ebendiese Module arbeiten möglicherweise problemlos auf anderen Plattformen, wo der voreingestellte Thread-Stack größer ist. Derartige Abstürze können Sie vermeiden, indem Sie `ThreadStackSize` auf einen höheren Wert als die Betriebssystemvoreinstellung setzen. Eine solche Anpassung ist nur notwendig, wenn es vom Anbieter des Moduls so spezifiziert wurde oder die Diagnose eines Apache-Absturzes ergeben hat, das die Thread-Stackgröße zu klein war.
- Auf Plattformen, wo die voreingestellte Thread-Stackgröße für die Webserverkonfiguration deutlich größer als notwendig ist, kann eine größere Anzahl von Threads pro Kindprozess erreicht werden, wenn

`ThreadStackSize` auf einen Wert kleiner als die Betriebssystemvoreinstellung gesetzt wird. Da es einzelne Anfragen geben kann, die mehr Stack zur Verarbeitung benötigen, sollte eine derartige Korrektur ausschließlich in Testumgebungen zum Einsatz kommen, auf denen die gesamte Webserververarbeitung ausprobiert werden kann. Eine Änderung der Webserverkonfiguration kann den aktuellen Wert der `ThreadStackSize` ungültig machen.



Beschreibung:	Die Benutzerkennung, unter welcher der Server Anfragen beantwortet
Syntax:	User <i>Unix-User-ID</i>
Voreinstellung:	User #-1
Kontext:	Serverkonfiguration
Status:	MPM
Modul:	leader , perchild , prefork , threadpool , worker
Kompatibilität:	Seit Apache 2.0 nur in der globalen Server-Konfiguration gültig

Die Direktive **User** legt die Benutzerkennung fest, mit der der Server Anfragen beantwortet. Um diese Anweisung zu verwenden, muss der Server als root gestartet werden. Wenn Sie den Server unter einem nicht-root-Benutzer starten, kann er nicht zu dem minder privilegierten Benutzer wechseln und wird statt dessen weiter mit der ursprünglichen Benutzerkennung laufen. Wenn Sie den Server als root starten, dann ist es normal, dass der Elternprozess als root weiterläuft. *Unix-User-ID* kann sein:

Ein Benutzername

Verweist auf den durch Namen angegebenen Benutzer.

gefolgt von einer Benutzernummer.

Verweist auf einen durch eine Nummer angegebenen Benutzer.

Der Benutzer sollte keine Rechte besitzen, die dazu führen, dass er in der Lage ist, auf Dateien zuzugreifen, die nicht dafür bestimmt sind, für die Außenwelt sichtbar zu sein. Gleichmaßen sollte der Benutzer nicht in der Lage sein, Code auszuführen, der nicht für HTTP-Anfragen bestimmt ist. Es wird empfohlen, einen neuen Benutzer und eine neue Gruppe speziell zur Ausführung des Servers zu erstellen. Einige Administratoren verwenden den Benutzer nobody. Dies ist jedoch nicht

immer wünschenswert, da der Benutzer nobody andere Rechte auf dem System besitzen kann.

Sicherheit

Setzen Sie **User** (oder **Group**) nicht auf root, solange Sie nicht genau wissen, was Sie tun, und welches die Gefahren sind.

Beim MPM **perchild**, das dafür gedacht ist, virtuelle Hosts unter verschiedenen Benutzerkennungen auszuführen, bestimmt die Direktive **User** die Benutzerkennung für den Hauptserver und bildet den Rückfallwert für **<VirtualHost>**-Abschnitte ohne eine **AssignUserID**-Anweisung.

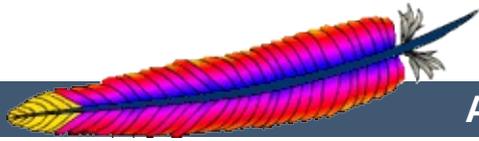
Wichtiger Hinweis: Die Verwendung dieser Direktive innerhalb von **<VirtualHost>** wird nicht mehr unterstützt. Benutzen Sie **SuexecUserGroup**, um Ihren Server für **suexec** einzurichten.

Anmerkung

Obwohl die Direktive **User** in den MPMs **beos** und **mpmt_os2** existiert, ist sie dort tatsächlich eine Leeranweisung und existiert nur aus Kompatibilitätsgründen.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Apache-MPM beos

<u>Beschreibung:</u>	Dieses Multi-Processing-Modul ist für BeOS optimiert.
<u>Status:</u>	MPM
<u>Modulbezeichner:</u>	mpm_beos_module
<u>Quelltext-Datei:</u>	beos.c

Zusammenfassung

Dieses Multi-Processing-Modul (MPM) ist das Standardmodul für BeOS. Es benutzt einen einzelnen Steuerprozess welcher Threads für die Bedienung der Anfragen erzeugt.

Siehe auch

[Adress- und Port-Einstellungen](#)



<u>Beschreibung:</u>	Die maximale Anzahl von Anfragen, die ein einzelner Thread während seiner Lebensdauer bedient.
<u>Syntax:</u>	MaxRequestsPerThread <i>Anzahl</i>
<u>Voreinstellung:</u>	MaxRequestsPerThread 0
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	beos

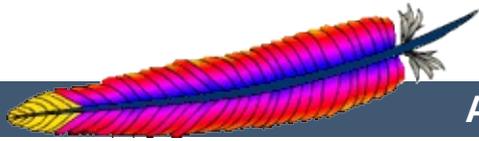
Die Direktive **MaxRequestsPerThread** legt die Anzahl der Anfragen fest, die ein einzelner Server-Thread bedient. Nach Erreichen der angegebenen Anzahl von Anfragen wird der Thread beendet. Wird für **MaxRequestsPerThread** der Wert 0 angegeben, wird der Thread niemals beendet.

Das Setzen von **MaxRequestsPerThread** auf einen Wert ungleich null hat zwei Vorteile:

- Die Menge des von einem Thread benötigten Speicherplatzes bei (unvorhergesehenen) Speicherproblemen kann begrenzt werden;
- Threads mit begrenzter Lebensdauer reduzieren die Anzahl der Threads bei reduzierter Serverlast.

Hinweis:

Bei **KeepAlive**-Anfragen wird nur die erste Anfrage auf das Maximum angerechnet. Das führt dazu, dass die Anzahl der *Verbindungen* pro Thread reduziert wird.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache MPM event

Description:	An experimental variant of the standard worker MPM
Status:	MPM
Module Identifier:	mpm_event_module
Source File:	event.c

Summary

Warning

This MPM is experimental, so it may or may not work as expected.

The [event](#) Multi-Processing Module (MPM) is designed to allow more requests to be served simultaneously by passing off some processing work to supporting threads, freeing up the main threads to work on new requests. It is based on the [worker](#) MPM, which implements a hybrid multi-process multi-threaded server. Run-time configuration directives are identical to those provided by [worker](#).

To use the [event](#) MPM, add `--with-mpm=event` to the [configure](#) script's arguments when building the [httpd](#).

See also

[The worker MPM](#)



HOW IT WORKS

This MPM tries to fix the 'keep alive problem' in HTTP. After a client completes the first request, the client can keep the connection open, and send further requests using the same socket. This can save significant overhead in creating TCP connections. However, Apache traditionally keeps an entire child process/thread waiting for data from the client, which brings its own disadvantages. To solve this problem, this MPM uses a dedicated thread to handle both the Listening sockets, and all sockets that are in a Keep Alive state.

The MPM assumes that the underlying `apr_pollset` implementation is reasonably threadsafe. This enables the MPM to avoid excessive high level locking, or having to wake up the listener thread in order to send it a keep-alive socket. This is currently only compatible with KQueue and EPoll.



Requirements

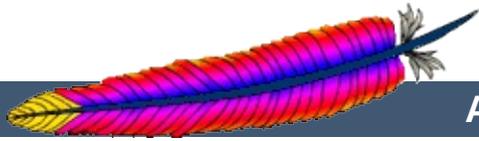
This MPM depends on [APR](#)'s atomic compare-and-swap operations for thread synchronization. If you are compiling for an x86 target and you don't need to support 386s, or you are compiling for a SPARC and you don't need to run on pre-UltraSPARC chips, add `--enable-nonportable-atomics=yes` to the [configure](#) script's arguments. This will cause APR to implement atomic operations using efficient opcodes not available in older CPUs.

This MPM does not perform well on older platforms which lack good threading, but the requirement for EPoll or KQueue makes this moot.

- To use this MPM on FreeBSD, FreeBSD 5.3 or higher is recommended. However, it is possible to run this MPM on FreeBSD 5.2.1, if you use `libkse` (see `man libmap.conf`).
- For NetBSD, at least version 2.0 is recommended.
- For Linux, a 2.6 kernel is recommended. It is also necessary to ensure that your version of `glibc` has been compiled with support for EPoll.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache MPM netware

Description:	Multi-Processing Module implementing an exclusively threaded web server optimized for Novell NetWare
Status:	MPM
Module Identifier:	mpm_netware_module
Source File:	mpm_netware.c

Summary

This Multi-Processing Module (MPM) implements an exclusively threaded web server that has been optimized for Novell NetWare.

The main thread is responsible for launching child worker threads which listen for connections and serve them when they arrive. Apache always tries to maintain several *spare* or idle worker threads, which stand ready to serve incoming requests. In this way, clients do not need to wait for a new child threads to be spawned before their requests can be served.

The [StartThreads](#), [MinSpareThreads](#), [MaxSpareThreads](#), and [MaxThreads](#) regulate how the main thread creates worker threads to serve requests. In general, Apache is very self-regulating, so most sites do not need to adjust these directives from their default values. Sites with limited memory may need to decrease [MaxThreads](#) to keep the server from thrashing (spawning and terminating idle threads). More information about tuning process creation is provided in the [performance hints](#) documentation.

[MaxRequestsPerChild](#) controls how frequently the server recycles processes by killing old ones and launching new ones. On the NetWare OS it is highly recommended that this directive remain set to 0. This allows worker threads to continue servicing requests

indefinitely.

See also

[Setting which addresses and ports Apache uses](#)



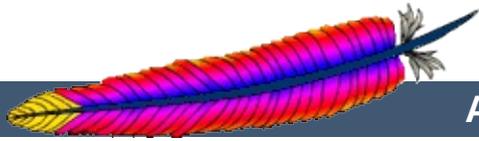
Description:	Set the maximum number of worker threads
Syntax:	MaxThreads <i>number</i>
Default:	MaxThreads 2048
Context:	server config
Status:	MPM
Module:	mpm_netware

The **MaxThreads** directive sets the desired maximum number worker threads allowable. The default value is also the compiled in hard limit. Therefore it can only be lowered, for example:

```
MaxThreads 512
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache MPM os2

Description:	Hybrid multi-process, multi-threaded MPM for OS/2
Status:	MPM
Module Identifier:	mpm_mpmt_os2_module
Source File:	mpmt_os2.c

Summary

The Server consists of a main, parent process and a small, static number of child processes.

The parent process's job is to manage the child processes. This involves spawning children as required to ensure there are always [StartServers](#) processes accepting connections.

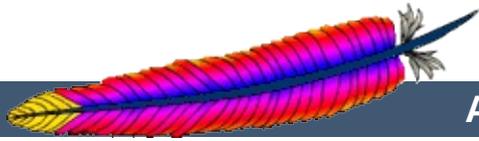
Each child process consists of a a pool of worker threads and a main thread that accepts connections and passes them to the workers via a work queue. The worker thread pool is dynamic, managed by a maintenance thread so that the number of idle threads is kept between [MinSpareThreads](#) and [MaxSpareThreads](#).

See also

[Setting which addresses and ports Apache uses](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Apache-MPM prefork

<u>Beschreibung:</u>	Implementiert einen im Voraus forkenden Webserver ohne Thread-Unterstützung
<u>Status:</u>	MPM
<u>Modulbezeichner:</u>	mpm_prefork_module
<u>Quelltext-Datei:</u>	prefork.c

Zusammenfassung

Dieses Multi-Processing-Modul (MPM) implementiert einen im Voraus forkenden Webserver ohne Thread-Unterstützung, der Anfragen auf ähnliche Weise behandelt wie der Apache 1.3. Es ist für Angebote geeignet, die aus Kompatibilitätsgründen mit nicht-Thread-sicheren Bibliotheken Threading vermeiden müssen. Es ist außerdem das geeignetste MPM, um jede Anfrage isoliert zu bearbeiten, so dass Probleme mit einem einzelnen Prozess keinen anderen beeinflussen.

Das MPM ist stark selbstregulierend, so dass es selten notwendig ist, seine Konfigurationseinstellungen zu justieren. Das Wichtigste ist, dass **MaxClients** gross genug ist, so viele gleichzeitige Anfragen zu bedienen, wie Sie erwarten, aber klein genug, um sicherzustellen, dass genug physischer Arbeitsspeicher für alle Prozesse vorhanden ist.

Siehe auch

[Bestimmen der vom Apache verwendeten Adressen und Ports](#)



Ein einzelner Steuerprozess ist für den Start von Kindprozessen verantwortlich, die auf Verbindungen warten und diese bedienen, sobald sie eintreffen. Der Apache versucht immer, mehrere *freie* oder unbeschäftigte Serverprozesse vorzuhalten, die zur Bedienung eingehender Anfragen bereit stehen. Auf diese Weise müssen Clients nicht darauf warten, dass neue Kindprozesse geforkt werden, bevor ihre Anfrage bearbeitet werden kann.

[StartServers](#), [MinSpareServers](#), [MaxSpareServers](#) und [MaxClients](#) regulieren, wie der Elternprozess Kindprozesse zur Bedienung von Anfragen erstellt. Im Allgemeinen ist der Apache sehr selbstregulierend, so dass die meisten Angebote die Voreinstellung dieser Direktiven nicht verändern müssen. Systeme, die mehr als 256 gleichzeitige Anfragen bedienen müssen, können [MaxClients](#) erhöhen, während Systeme mit begrenztem Arbeitsspeicher möglicherweise [MaxClients](#) heruntersetzen müssen, um den Server vor Flatterverhalten (Arbeitsspeicherinhalte auf Platte auslagern - und zurück) zu schützen. Weitere Informationen zur Feinabstimmung der Prozesserstellung sind in den [Performance-Hinweisen](#) zu finden.

Während der Elternprozess unter Unix normalerweise als root gestartet wird, um sich an Port 80 binden zu können, werden die Kindprozesse unter einem weniger privilegierten Benutzer gestartet. Die Direktiven [User](#) und [Group](#) werden dazu verwendet, die Privilegien der Apache-Kindprozesse festzulegen. Die Kindprozesse müssen in der Lage sein, alle Inhalte zu lesen, die sie ausliefern sollen, sollten darüber hinaus jedoch so wenig wie möglich Rechte besitzen.

[MaxRequestsPerChild](#) bestimmt, wie häufig der Server Prozesse erneuert, indem er alte beendet und neue startet.



<u>Beschreibung:</u>	Maximale Anzahl der unbeschäftigten Kindprozesse des Servers
<u>Syntax:</u>	MaxSpareServers <i>Anzahl</i>
<u>Voreinstellung:</u>	MaxSpareServers 10
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	prefork

Die Direktive [MaxSpareServers](#) bestimmt das gewünschte Maximum an *unbeschäftigten* Kindprozessen des Servers. Ein unbeschäftigter Prozess ist einer, der keine Anfrage bedient. Wenn mehr als [MaxSpareServers](#) Prozesse unbeschäftigt sind, wird der Elternprozess die überschüssigen Prozesse beenden.

Eine Feineinstellung dieses Parameters sollte nur bei sehr beschäftigten Angeboten notwendig sein. Es ist nahezu immer eine schlechte Idee, den Parameter auf einen hohen Wert zu setzen. Wenn Sie versuchen, den Wert kleiner oder gleich [MinSpareServers](#) zu setzen, wird der Apache ihn automatisch auf [MinSpareServers](#) + 1 korrigieren.

Siehe auch

- [MinSpareServers](#)
- [StartServers](#)



<u>Beschreibung:</u>	Minimale Anzahl der unbeschäftigten Kindprozesse des Servers
<u>Syntax:</u>	<code>MinSpareServers <i>Anzahl</i></code>
<u>Voreinstellung:</u>	<code>MinSpareServers 5</code>
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	prefork

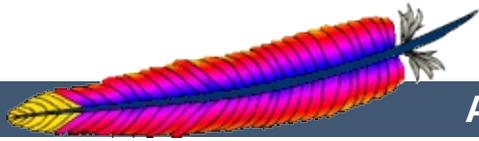
Die Direktive `MinSpareServers` bestimmt das gewünschte Minimum der *unbeschäftigten* Kindprozesse des Servers. Ein unbeschäftigter Prozess ist einer, der keine Anfrage bedient. Wenn weniger als `MinSpareServers` Prozesse unbeschäftigt sind, dann erstellt der Elternprozess neue mit einer maximalen Rate von 1 pro Sekunde.

Die Feineinstellung des Parameters sollte nur bei sehr beschäftigten Angeboten notwendig sein. Es ist nahezu immer eine schlechte Idee, den Parameter auf einen hohen Wert zu setzen.

Siehe auch

- [MaxSpareServers](#)
- [StartServers](#)

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Apache-MPM winnt

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

<u>Beschreibung:</u>	Das Multi-Processing-Modul ist optimiert für Windows NT.
<u>Status:</u>	MPM
<u>Modulbezeichner:</u>	mpm_winnt_module
<u>Quelltext-Datei:</u>	mpm_winnt.c

Zusammenfassung

Dieses Multi-Processing-Modul (MPM) ist die Voreinstellung für das Betriebssystem Windows NT. Es verwendet einen einzelnen Steuerprozess, der einen einzelnen Kindprozess startet, welcher wiederum Threads zur Bedienung von Anfragen erstellt.



<u>Beschreibung:</u>	Für die Annahme von Netzwerkverbindungen wird <code>accept()</code> anstelle von <code>AcceptEx()</code> verwendet
<u>Syntax:</u>	<code>Win32DisableAcceptEx</code>
<u>Voreinstellung:</u>	<code>AcceptEx()</code> ist standardmäßig aktiviert. Verwenden Sie diese Direktive, um den Gebrauch von <code>AcceptEx()</code> zu deaktivieren.
<u>Kontext:</u>	Serverkonfiguration
<u>Status:</u>	MPM
<u>Modul:</u>	<code>mpm_winnt</code>
<u>Kompatibilität:</u>	Verfügbar ab Version 2.0.49

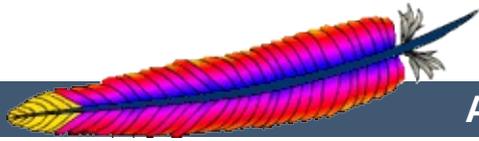
`AcceptEx()` ist eine Schnittstelle zu Microsoft Winsock v2, die unter bestimmten Umständen einige Leistungsverbesserungen gegenüber der `accept()`-API von BSD bietet. Einige beliebte Windows-Produkte, typischerweise Virens Scanner oder VPN-Pakete, besitzen jedoch Fehler, welche den einwandfreien Betrieb von `AcceptEx()` stören. Wenn Sie einen Fehler wie:

```
[error] (730038)An operation was attempted on something that is not a socket.: winnt_accept: AcceptEx failed. Attempting to recover.
```

erhalten, sollten Sie diese Direktive verwenden, um den Gebrauch von `AcceptEx()` zu unterbinden.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Apache-MPM worker

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

<u>Beschreibung:</u>	Multi-Processing-Modul, das einen Hybrid-Webserver mit Multi-Thread und Multi-Prozess-Unterstützung implementiert
<u>Status:</u>	MPM
<u>Modulbezeichner:</u>	mpm_worker_module
<u>Quelltext-Datei:</u>	worker.c

Zusammenfassung

Dieses Multi-Processing-Modul (MPM) implementiert einen Hybrid-Server mit Multi-Thread und Multi-Prozess-Unterstützung. Durch die Verwendung von Threads für die Bedienung von Anfragen ist er in der Lage, eine große Anzahl von Anfragen mit weniger Systemressourcen als ein Prozess-basierter Server zu bedienen. Er behält jedoch viel von der Stabilität eines Prozess-basierten Servers bei, indem er mehrere Prozesse verfügbar hält, jeden mit etlichen Threads.

Die wichtigsten Direktiven zur Steuerung des MPMs sind [ThreadsPerChild](#), welche die Anzahl der Threads beeinflusst, die von jedem Kindprozess verwendet werden, und [MaxClients](#), welche die maximale Gesamtzahl an Threads regelt, die gestartet werden können.

Siehe auch

[Bestimmen der vom Apache verwendeten Adressen und Ports](#)



Ein einzelner Steuerprozess (der Elternprozess) ist für den Start der Kindprozesse verantwortlich. Jeder Kindprozess erstellt eine feste Anzahl von Server-Threads, wie durch die `ThreadsPerChild`-Direktive angegeben, sowie einen "Listener-Thread", der auf Verbindungen wartet und diese an einen Server-Thread zur Bearbeitung weiterreicht, sobald sie eintreffen.

Der Apache versucht immer, einen Vorrat von *freien* oder unbeschäftigten Threads zu verwalten, die zur Bedienung hereinkommender Anfragen bereit stehen. Auf diese Weise brauchen Clients nicht auf die Erstellung eines neuen Threads oder Prozesses zu warten, bevor ihre Anfrage bedient werden kann. Die Anzahl der Prozesse, die anfangs gestartet wird, wird mit der Direktive `StartServers` festgelegt. Dann, während des Betriebes, berechnet der Apache die Gesamtzahl der unbeschäftigten Threads und forkt oder beendet Prozesse, um diese Anzahl innerhalb der durch `MinSpareThreads` und `MaxSpareThreads` angegebenen Grenzen zu halten. Da dieser Prozess sehr selbstregulierend ist, ist es nur selten notwendig, die Voreinstellung dieser Direktiven zu ändern. Die maximale Anzahl Clients, die gleichzeitig bedient werden kann (d.h. die maximale Gesamtzahl der Threads in allen Prozessen), wird mit der Direktive `MaxClients` festgelegt. Die maximale Anzahl der aktiven Kindprozesse ergibt sich aus `MaxClients` dividiert durch `ThreadsPerChild`.

Zwei Direktiven legen harte Limits für die Anzahl der aktiven Kindprozesse fest und können nur geändert werden, indem der Server komplett gestoppt und dann wieder neu gestartet wird. `ServerLimit` stellt die obere Grenze für die Anzahl der aktiven Kindprozesse dar und muss größer oder gleich dem Quotienten aus `MaxClients` und `ThreadsPerChild` sein. `ThreadLimit` ist die obere Grenze für die Anzahl der Server-Threads und muss größer oder gleich `ThreadsPerChild` sein. Sofern für diese Direktiven keine Voreinstellungen verwendet werden, sollten sie vor allen anderen

worker-Direktiven platziert werden.

Neben den normalen aktiven Kindprozessen gibt es möglicherweise noch zusätzliche Kindprozesse, welche gerade beendet werden, wo allerdings zumindest noch ein Server-Thread eine existierende Verbindung bearbeitet. Obwohl die tatsächlich zu erwartende Anzahl deutlich kleiner ist, können bis zu MaxClients solcher Prozesse auftreten. Dieses Verhalten können Sie vermeiden, indem Sie die Terminierung einzelner Kindprozesse wie folgt abschalten:

- setzen Sie den Wert von MaxRequestsPerChild auf Null
- setzen Sie den Wert von MaxSpareThreads auf den gleichen Wert wie MaxClients

Eine typische Konfiguration der Prozess-Thread-Steuerung für das MPM worker könnte wie folgt aussehen:

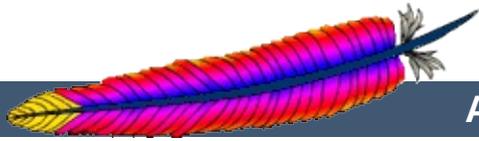
```
ServerLimit 16
StartServers 2
MaxClients 150
MinSpareThreads 25
MaxSpareThreads 75
ThreadsPerChild 25
```

Während der Elternprozess unter Unix normalerweise als root gestartet wird, um sich an Port 80 binden zu können, werden die Kindprozesse und Threads unter einem weniger privilegierten Benutzer gestartet. Die Direktiven User und Group werden dazu verwendet, die Privilegien der Apache-Kindprozesse festzulegen. Die Kindprozesse müssen in der Lage sein, alle Inhalte zu lesen, die sie ausliefern sollen, sollten darüber hinaus jedoch so wenig wie möglich Rechte besitzen. Zusätzlich, solange nicht suexec verwendet wird, legen diese Direktiven auch die Privilegien fest, die von CGI-Skripts geerbt werden.

MaxRequestsPerChild bestimmt, wie häufig der Server Prozesse erneuert, indem er alte beendet und neue startet.

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Apache-Modul mod_actions

<u>Beschreibung:</u>	Dieses Modul ermöglicht die Ausführung von CGI-Skripten in Abhängigkeit von Medientypen und Anfragemethoden.
<u>Status:</u>	Basis
<u>Modulbezeichner:</u>	actions_module
<u>Quelltext-Datei:</u>	mod_actions.c

Zusammenfassung

Das Modul besitzt zwei Direktiven. Die Direktive [Action](#) erlaubt die Ausführung von CGI-Skripten immer dann, wenn eine Anfrage zu einem bestimmten [MIME-Type](#) erfolgt. Die Direktive [Script](#) erlaubt die Ausführung von CGI-Skripten abhängig von einer bestimmten Methode, die in der Anfrage verwendet wird. Dies macht es deutlich einfacher, Skripte auszuführen, die Dateien verarbeiten.

Siehe auch

[mod_cgi](#)

[Dynamische Inhalte mit CGI](#)

[Die Verwendung von Handlern](#)



<u>Beschreibung:</u>	Aktiviert ein CGI-Skript für einen bestimmten Handler oder Content-Type
<u>Syntax:</u>	Action <i>Aktionsart</i> <i>CGI-Skript</i> [virtual]
<u>Kontext:</u>	Serverkonfiguration, Virtual Host, Verzeichnis, .htaccess
<u>AllowOverride:</u>	FileInfo
<u>Status:</u>	Basis
<u>Modul:</u>	mod_actions
<u>Kompatibilität:</u>	Der Schalter <code>virtual</code> und die Übergabe des Handlers wurden in Apache 2.1 eingeführt.

Die Direktive fügt eine Aktion hinzu, welche das *CGI-Skript* aktiviert, sobald die *Aktionsart* durch eine Anfrage ausgelöst wird. *CGI-Skript* ist der URL-Pfad zu einer Ressource, die unter Verwendung von [ScriptAlias](#) oder [AddHandler](#) als CGI-Skript gekennzeichnet wurde. Die *Aktionsart* kann entweder ein [Handler](#) oder ein [MIME-Type](#) sein. Die URL und den Dateipfad des angeforderten Dokuments in den Standard-CGI-Umgebungsvariablen `PATH_INFO` und `PATH_TRANSLATED` übergeben. Der für die jeweilige Anfrage verwendete Handler wird in der Umgebungsvariablen `REDIRECT_HANDLER` übergeben.

Beispiele

```
# Anfragen für Dateien eines bestimmten MIME-Types:
Action image/gif /cgi-bin/images.cgi

# Dateien einer bestimmten Dateiendung
AddHandler my-file-type .xyz
Action my-file-type /cgi-bin/program.cgi
```

Im ersten Beispiel werden Anfragen für Dateien mit dem MIME-Type `image/gif` von dem angegebenen CGI-Skript `/cgi-bin/images.cgi` bearbeitet.

Im zweiten Beispiel werden Anfragen für Dateien mit der Dateierdung `.xyz` von dem angegebenen CGI-Skript `/cgi-bin/program.cgi` bearbeitet.

Der optionale Schalter `virtual` deaktiviert die Prüfung auf Existenz der angeforderten Datei. Dies ist beispielsweise nützlich, wenn Sie die Direktive `Action` in Verbindung mit virtuellen Adressräumen verwenden möchten.

Beispiel

```
<Location /news>
  SetHandler news-handler
  Action news-handler /cgi-bin/news.cgi virtual
</Location>
```

Siehe auch

- [AddHandler](#)



Beschreibung:	Aktiviert ein CGI-Skript für eine bestimmte Anfragemethode.
Syntax:	<code>Script Methode CGI-Skript</code>
Kontext:	Serverkonfiguration, Virtual Host, Verzeichnis
Status:	Basis
Modul:	mod_actions

Die Direktive fügt eine Aktion hinzu, welche das *CGI-Skript* aktiviert, wenn eine Datei unter der Verwendung der Methode *Methode* angefordert wird. *CGI-Skript* ist der URL-Pfad zu einer Ressource, die unter Verwendung von [ScriptAlias](#) oder [AddHandler](#) als CGI-Skript gekennzeichnet wurde. Die URL und der Dateipfad des angeforderten Dokuments werden in den Standard-CGI-Umgebungsvariablen `PATH_INFO` und `PATH_TRANSLATED` übergeben.

Der Methodename kann frei gewählt werden. Bei **Methodennamen** wird zwischen **Groß- und Kleinschreibung** unterschieden, so dass `Script PUT` und `Script put` zu vollkommen unterschiedlichen Ergebnissen führen.

Beachten Sie, dass der `Script`-Befehl nur Voreinstellungen für Aktionen definiert. Wird ein CGI-Skript - oder eine andere Ressource, die in der Lage ist, die angeforderte Methode intern zu bearbeiten - aufgerufen, so wird diese(s) verwendet. Beachten Sie auch, dass `Script` mit der Methode GET nur dann aufgerufen wird, wenn Query-Argumente vorhanden sind (z.B. `foo.html?hi`). Andernfalls wird die Anfrage normal bearbeitet.

Beispiele

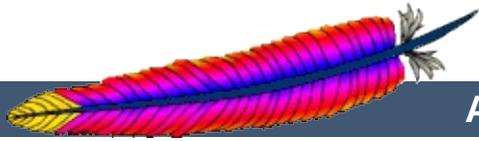
```
# Für <ISINDEX>-ähnliches Suchen
Script GET /cgi-bin/search
```

```
# Ein CGI-PUT-Handler
```

Script PUT /~bob/put.cgi

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_alias`

Description:	Provides for mapping different parts of the host filesystem in the document tree and for URL redirection
Status:	Base
Module Identifier:	<code>alias_module</code>
Source File:	<code>mod_alias.c</code>

Summary

The directives contained in this module allow for manipulation and control of URLs as requests arrive at the server. The [Alias](#) and [ScriptAlias](#) directives are used to map between URLs and filesystem paths. This allows for content which is not directly under the [DocumentRoot](#) served as part of the web document tree. The [ScriptAlias](#) directive has the additional effect of marking the target directory as containing only CGI scripts.

The [Redirect](#) directives are used to instruct clients to make a new request with a different URL. They are often used when a resource has moved to a new location.

[mod_alias](#) is designed to handle simple URL manipulation tasks. For more complicated tasks such as manipulating the query string, use the tools provided by [mod_rewrite](#).

See also

[mod_rewrite](#)

[Mapping URLs to the filesystem](#)



Aliases and Redirects occurring in different contexts are processed like other directives according to standard [merging rules](#). But when multiple Aliases or Redirects occur in the same context (for example, in the same [<VirtualHost>](#) section) they are processed in a particular order.

First, all Redirects are processed before Aliases are processed, and therefore a request that matches a [Redirect](#) or [RedirectMatch](#) will never have Aliases applied. Second, the Aliases and Redirects are processed in the order they appear in the configuration files, with the first match taking precedence.

For this reason, when two or more of these directives apply to the same sub-path, you must list the most specific path first in order for all the directives to have an effect. For example, the following configuration will work as expected:

```
Alias /foo/bar /baz
Alias /foo /gaq
```

But if the above two directives were reversed in order, the `/foo` [Alias](#) would always match before the `/foo/bar` [Alias](#), so the latter directive would be ignored.



Description:	Maps URLs to filesystem locations
Syntax:	Alias <i>URL-path file-path directory-path</i>
Context:	server config, virtual host
Status:	Base
Module:	mod_alias

The [Alias](#) directive allows documents to be stored in the local filesystem other than under the [DocumentRoot](#). URLs with a (%-decoded) path beginning with *URL-path* will be mapped to local files beginning with *directory-path*. The *URL-path* is case-sensitive, even on case-insensitive file systems.

Example:

```
Alias /image /ftp/pub/image
```

A request for `http://example.com/image/foo.gif` would cause the server to return the file `/ftp/pub/image/foo.gif`. Only complete path segments are matched, so the above alias would not match a request for `http://example.com/imagefoo.gif`. For more complex matching using regular expressions, see the [AliasMatch](#) directive.

Note that if you include a trailing `/` on the *URL-path* then the server will require a trailing `/` in order to expand the alias. That is, if you use

```
Alias /icons/ /usr/local/apache/icons/
```

then the URL `/icons` will not be aliased, as it lacks that trailing `/`. Likewise, if you omit the slash on the *URL-path* then you must also omit it from the *file-path*.

Note that you may need to specify additional [<Directory>](#) sections which cover the *destination* of aliases. Aliasing occurs before

`<Directory>` sections are checked, so only the destination of aliases are affected. (Note however `<Location>` sections are run through once before aliases are performed, so they will apply.)

In particular, if you are creating an Alias to a directory outside of your `DocumentRoot`, you may need to explicitly permit access to the target directory.

Example:

```
Alias /image /ftp/pub/image
<Directory /ftp/pub/image>
  Order allow,deny
  Allow from all
</Directory>
```

Any number of slashes in the *URL-path* parameter matches any number of slashes in the requested URL-path.



Description:	Maps URLs to filesystem locations using regular expressions
Syntax:	<code>AliasMatch <i>regex file-path directory-path</i></code>
Context:	server config, virtual host
Status:	Base
Module:	<code>mod_alias</code>

This directive is equivalent to [Alias](#), but makes use of [regular expressions](#), instead of simple prefix matching. The supplied regular expression is matched against the URL-path, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a filename. For example, to activate the `/icons` directory, one might use:

```
AliasMatch ^/icons(.*) /usr/local/apache/icons$1
```

The full range of [regular expression](#) power is available. For example, it is possible to construct an alias with case-insensitive matching of the URL-path:

```
AliasMatch (?i)~/image(.*) /ftp/pub/image$1
```

One subtle difference between [Alias](#) and [AliasMatch](#) is that [Alias](#) will automatically copy any additional part of the URI, past the part that matched, onto the end of the file path on the right side, while [AliasMatch](#) will not. This means that in almost all cases, you will want the regular expression to match the entire request URI from beginning to end, and to use substitution on the right side.

In other words, just changing [Alias](#) to [AliasMatch](#) will not have the same effect. At a minimum, you need to add `^` to the beginning of the regular expression and add `(.*)$` to the end, and add `$1` to the end of

the replacement.

For example, suppose you want to replace this with AliasMatch:

```
Alias /image/ /ftp/pub/image/
```

This is NOT equivalent - don't do this! This will send all requests that have /image/ anywhere in them to /ftp/pub/image/:

```
AliasMatch /image/ /ftp/pub/image/
```

This is what you need to get the same effect:

```
AliasMatch ^/image/(.*)$ /ftp/pub/image/$1
```

Of course, there's no point in using [AliasMatch](#) where [Alias](#) would work. [AliasMatch](#) lets you do more complicated things. For example, you could serve different kinds of files from different directories:

```
AliasMatch ^/image/(.*)\.jpg$ /files/jpg.images/$1.jpg  
AliasMatch ^/image/(.*)\.gif$ /files/gif.images/$1.gif
```

Multiple leading slashes in the requested URL are discarded by the server before directives from this module compares against the requested URL-path.



Description:	Sends an external redirect asking the client to fetch a different URL
Syntax:	Redirect [<i>status</i>] <i>URL-path</i> <i>URL</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias
Compatibility:	Support for specifying a URL-path as the final argument was added in Apache HTTP Server 2.2.6

The Redirect directive maps an old URL into a new one by asking the client to refetch the resource at the new location.

The old *URL-path* is a case-sensitive (%-decoded) path beginning with a slash. A relative path is not allowed. The new *URL* should be an absolute URL beginning with a scheme and hostname. In Apache HTTP Server 2.2.6 and later, a URL-path beginning with a slash may also be used, in which case the scheme and hostname of the current server will be added.

Then any request beginning with *URL-path* will return a redirect request to the client at the location of the target *URL*. Additional path information beyond the matched *URL-path* will be appended to the target URL.

Example:

```
Redirect /service http://foo2.example.com/service
```

If the client requests `http://example.com/service/foo.txt`, it will be told to access `http://foo2.example.com/service/foo.txt` instead. Only complete path segments are matched, so the above example would not match a request for `http://example.com/servicefoo.txt`. For more complex

matching using regular expressions, see the [RedirectMatch](#) directive.

Note

Redirect directives take precedence over `Alias` and `ScriptAlias` directives, irrespective of their ordering in the configuration file.

If no *status* argument is given, the redirect will be "temporary" (HTTP status 302). This indicates to the client that the resource has moved temporarily. The *status* argument can be used to return other HTTP status codes:

permanent

Returns a permanent redirect status (301) indicating that the resource has moved permanently.

temp

Returns a temporary redirect status (302). This is the default.

seeother

Returns a "See Other" status (303) indicating that the resource has been replaced.

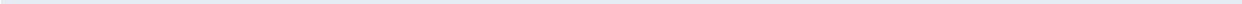
gone

Returns a "Gone" status (410) indicating that the resource has been permanently removed. When this status is used the *URL* argument should be omitted.

Other status codes can be returned by giving the numeric status code as the value of *status*. If the status is between 300 and 399, the *URL* argument must be present, otherwise it must be omitted. Note that the status must be known to the Apache code (see the function `send_error_response` in `http_protocol.c`).

Example:

```
Redirect permanent /one http://example.com/two
Redirect 303 /three http://example.com/other
```



Description:	Sends an external redirect based on a regular expression match of the current URL
Syntax:	RedirectMatch [<i>status</i>] <i>regex URL</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias

This directive is equivalent to [Redirect](#), but makes use of [regular expressions](#), instead of simple prefix matching. The supplied regular expression is matched against the URL-path, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a filename. For example, to redirect all GIF files to like-named JPEG files on another server, one might use:

```
RedirectMatch (.*)\.gif$ http://www.anotherserver.com$1.jpg
```



Description:	Sends an external permanent redirect asking the client to fetch a different URL
Syntax:	RedirectPermanent <i>URL-path URL</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias

This directive makes the client know that the Redirect is permanent (status 301). Exactly equivalent to Redirect permanent.



RedirectTemp Directive

Description:	Sends an external temporary redirect asking the client to fetch a different URL
Syntax:	RedirectTemp <i>URL -path URL</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias

This directive makes the client know that the Redirect is only temporary (status 302). Exactly equivalent to Redirect temp.



Description: Maps a URL to a filesystem location and designates the target as a CGI script

Syntax: `ScriptAlias URL-path file-path|directory-path`

Context: server config, virtual host

Status: Base

Module: mod_alias

The `ScriptAlias` directive has the same behavior as the `Alias` directive, except that in addition it marks the target directory as containing CGI scripts that will be processed by `mod_cgi`'s `cgi-script` handler. URLs with a case-sensitive (%-decoded) path beginning with `URL-path` will be mapped to scripts beginning with the second argument, which is a full pathname in the local filesystem.

Example:

```
ScriptAlias /cgi-bin/ /web/cgi-bin/
```

A request for `http://example.com/cgi-bin/foo` would cause the server to run the script `/web/cgi-bin/foo`. This configuration is essentially equivalent to:

```
Alias /cgi-bin/ /web/cgi-bin/  
<Location /cgi-bin >  
    SetHandler cgi-script  
    Options +ExecCGI  
</Location>
```

It is safer to avoid placing CGI scripts under the `DocumentRoot` in order to avoid accidentally revealing their source code if the configuration is ever changed. The `ScriptAlias` makes this easy by mapping a URL and designating CGI scripts at the same time. If you do choose to place your CGI scripts in a directory already accessible

from the web, do not use `ScriptAlias`. Instead, use `<Directory>`, `SetHandler`, and `Options` as in:

```
<Directory /usr/local/apache2/htdocs/cgi-bin >  
  SetHandler cgi-script  
  Options ExecCGI  
</Directory>
```

This is necessary since multiple *URL-paths* can map to the same filesystem location, potentially bypassing the `ScriptAlias` and revealing the source code of the CGI scripts if they are not restricted by a `Directory` section.

See also

- [CGI Tutorial](#)



Description:	Maps a URL to a filesystem location using a regular expression and designates the target as a CGI script
Syntax:	<code>ScriptAliasMatch <i>regex file-path directory-path</i></code>
Context:	server config, virtual host
Status:	Base
Module:	mod_alias

This directive is equivalent to [ScriptAlias](#), but makes use of [regular expressions](#), instead of simple prefix matching. The supplied regular expression is matched against the URL-path, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a filename. For example, to activate the standard /cgi-bin, one might use:

```
ScriptAliasMatch ^/cgi-bin(.*) /usr/local/apache/cgi-bin$1
```

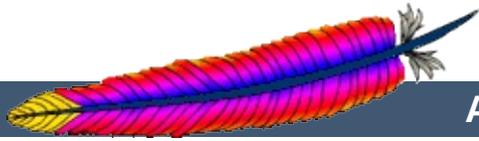
As for [AliasMatch](#), the full range of [regular expression](#) power is available. For example, it is possible to construct an alias with case-insensitive matching of the URL-path:

```
ScriptAliasMatch (?i)^/cgi-bin(.*) /usr/local/apache/cgi-bin$1
```

The considerations related to the difference between [Alias](#) and [AliasMatch](#) also apply to the difference between [ScriptAlias](#) and [ScriptAliasMatch](#). See [AliasMatch](#) for details.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_asis`

Description:	Sends files that contain their own HTTP headers
Status:	Base
Module Identifier:	<code>asis_module</code>
Source File:	<code>mod_asis.c</code>

Summary

This module provides the handler `send-as-is` which causes Apache to send the document without adding most of the usual HTTP headers.

This can be used to send any kind of data from the server, including redirects and other special HTTP responses, without requiring a cgi-script or an nph script.

For historical reasons, this module will also process any file with the mime type `httpd/send-as-is`.

See also

[mod_headers](#)

[mod_cern_meta](#)

[Apache's Handler Use](#)



In the server configuration file, associate files with the `send-as-is` handler e.g.

```
AddHandler send-as-is asis
```

The contents of any file with a `.asis` extension will then be sent by Apache to the client with almost no changes. In particular, HTTP headers are derived from the file itself according to [mod_cgi](#) rules, so an `asis` file must include valid headers, and may also use the `CGI Status:` header to determine the HTTP response code.

Here's an example of a file whose contents are sent *as is* so as to tell the client that a file has redirected.

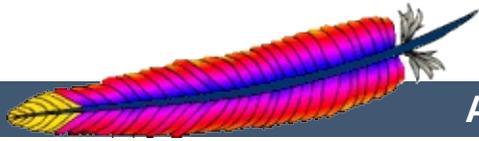
```
Status: 301 Now where did I leave that URL
Location: http://xyz.abc.com/foo/bar.html
Content-type: text/html

<html>
<head>
<title>Lame excuses'R'us</title>
</head>
<body>
<h1>Fred's exceptionally wonderful page has moved to
<a href="http://xyz.abc.com/foo/bar.html">Joe's</a> site.
</h1>
</body>
</html>
```

Notes:

The server always adds a `Date:` and `Server:` header to the data returned to the client, so these should not be included in the file. The server does *not* add a `Last-Modified` header; it probably should.

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_auth_basic`

Description:	Basic authentication
Status:	Base
Module Identifier:	<code>auth_basic_module</code>
Source File:	<code>mod_auth_basic.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module allows the use of HTTP Basic Authentication to restrict access by looking up users in the given providers. HTTP Digest Authentication is provided by [mod_auth_digest](#). This module should usually be combined with at least one authentication module such as [mod_authn_file](#) and one authorization module such as [mod_authz_user](#).

See also

[AuthName](#)

[AuthType](#)

[Require](#)

[Satisfy](#)

[Authentication howto](#)



Description:	Sets whether authorization and authentication are passed to lower level modules
Syntax:	AuthBasicAuthoritative On Off
Default:	AuthBasicAuthoritative On
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	mod_auth_basic

Normally, each authorization module listed in [AuthBasicProvider](#) will attempt to verify the user, and if the user is not found in any provider, access will be denied. Setting the [AuthBasicAuthoritative](#) directive explicitly to `Off` allows for both authentication and authorization to be passed on to other non-provider-based modules if there is **no userID** or **rule** matching the supplied userID. This should only be necessary when combining [mod_auth_basic](#) with third-party modules that are not configured with the [AuthBasicProvider](#) directive. When using such modules, the order of processing is determined in the modules' source code and is not configurable.



Description:	Sets the authentication provider(s) for this location
Syntax:	AuthBasicProvider <i>provider-name</i> [<i>provider-name</i>] ...
Default:	AuthBasicProvider file
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	mod_auth_basic

The `AuthBasicProvider` directive sets which provider is used to authenticate the users for this location. The default `file` provider is implemented by the `mod_authn_file` module. Make sure that the chosen provider module is present in the server.

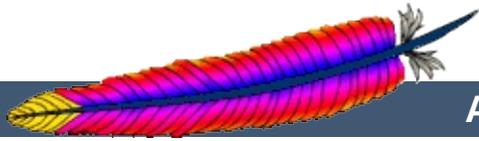
Example

```
<Location /secure>
  AuthType basic
  AuthName "private area"
  AuthBasicProvider dbm
  AuthDBMType SDBM
  AuthDBMUserFile /www/etc/dbmpasswd
  Require valid-user
</Location>
```

Providers are queried in order until a provider finds a match for the requested username, at which point this sole provider will attempt to check the password. A failure to verify the password does not result in control being passed on to subsequent providers.

Providers are implemented by `mod_authn_dbm`, `mod_authn_file`, `mod_authn_dbd`, and `mod_authnz_ldap`.

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_auth_digest`

Description:	User authentication using MD5 Digest Authentication.
Status:	Extension
Module Identifier:	<code>auth_digest_module</code>
Source File:	<code>mod_auth_digest.c</code>

Summary

This module implements HTTP Digest Authentication ([RFC2617](#)), and provides an alternative to `mod_auth_basic` where the password is not transmitted as cleartext. However, this does **not** lead to a significant security advantage over basic authentication. On the other hand, the password storage on the server is much less secure with digest authentication than with basic authentication. Therefore, using basic auth and encrypting the whole connection using `mod_ssl` is a much better alternative.

See also

[AuthName](#)

[AuthType](#)

[Require](#)

[Satisfy](#)

[Authentication howto](#)



Using Digest Authentication

Using MD5 Digest authentication is very simple. Simply set up authentication normally, using `AuthType Digest` and [AuthDigestProvider](#) instead of the normal `AuthType Basic` and [AuthBasicProvider](#). Then add a [AuthDigestDomain](#) directive containing at least the root URI(s) for this protection space.

Appropriate user (text) files can be created using the [htdigest](#) tool.

Example:

```
<Location /private/>
  AuthType Digest
  AuthName "private area"
  AuthDigestDomain /private/ http://mirror.my.dom/private2/

  AuthDigestProvider file
  AuthUserFile /web/auth/.digest_pw
  Require valid-user
</Location>
```

Note

Digest authentication was intended to be more secure than basic authentication, but no longer fulfills that design goal. A man-in-the-middle attacker can trivially force the browser to downgrade to basic authentication. And even a passive eavesdropper can brute-force the password using today's graphics hardware, because the hashing algorithm used by digest authentication is too fast. Another problem is that the storage of the passwords on the server is insecure. The contents of a stolen `htdigest` file can be used directly for digest authentication. Therefore using [mod_ssl](#) to encrypt the whole connection is strongly recommended.



Description:	Selects the algorithm used to calculate the challenge and response hashes in digest authentication
Syntax:	<code>AuthDigestAlgorithm MD5 MD5-sess</code>
Default:	<code>AuthDigestAlgorithm MD5</code>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_digest

The `AuthDigestAlgorithm` directive selects the algorithm used to calculate the challenge and response hashes.

MD5 - sess is not correctly implemented yet.



Description:	URIs that are in the same protection space for digest authentication
Syntax:	AuthDigestDomain <i>URI</i> [<i>URI</i>] ...
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_digest

The `AuthDigestDomain` directive allows you to specify one or more URIs which are in the same protection space (*i.e.* use the same realm and username/password info). The specified URIs are prefixes; the client will assume that all URIs "below" these are also protected by the same username/password. The URIs may be either absolute URIs (*i.e.* including a scheme, host, port, etc.) or relative URIs.

This directive *should* always be specified and contain at least the (set of) root URI(s) for this space. Omitting to do so will cause the client to send the Authorization header for *every request* sent to this server. Apart from increasing the size of the request, it may also have a detrimental effect on performance if `AuthDigestNcCheck` is on.

The URIs specified can also point to different servers, in which case clients (which understand this) will then share username/password info across multiple servers without prompting the user each time.



Description:	Enables or disables checking of the nonce-count sent by the server
Syntax:	AuthDigestNcCheck On Off
Default:	AuthDigestNcCheck Off
Context:	server config
Status:	Extension
Module:	mod_auth_digest

Not implemented yet.



Description:	Determines how the nonce is generated
Syntax:	AuthDigestNonceFormat <i>format</i>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_digest

Not implemented yet.



Description:	How long the server nonce is valid
Syntax:	<code>AuthDigestNonceLifetime seconds</code>
Default:	<code>AuthDigestNonceLifetime 300</code>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_digest

The `AuthDigestNonceLifetime` directive controls how long the server nonce is valid. When the client contacts the server using an expired nonce the server will send back a 401 with `stale=true`. If `seconds` is greater than 0 then it specifies the amount of time for which the nonce is valid; this should probably never be set to less than 10 seconds. If `seconds` is less than 0 then the nonce never expires.



Description:	Sets the authentication provider(s) for this location
Syntax:	<code>AuthDigestProvider <i>provider-name</i></code> <code>[<i>provider-name</i>] ...</code>
Default:	<code>AuthDigestProvider file</code>
Context:	directory, .htaccess
Override:	<code>AuthConfig</code>
Status:	Extension
Module:	<code>mod_auth_digest</code>

The `AuthDigestProvider` directive sets which provider is used to authenticate the users for this location. The default `file` provider is implemented by the [mod_authn_file](#) module. Make sure that the chosen provider module is present in the server.

See [mod_authn_dbm](#), [mod_authn_file](#), and [mod_authn_dbd](#) for providers.



Description:	Determines the quality-of-protection to use in digest authentication
Syntax:	AuthDigestQop none auth auth-int [auth auth-int]
Default:	AuthDigestQop auth
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_digest

The `AuthDigestQop` directive determines the *quality-of-protection* to use. `auth` will only do authentication (username/password); `auth-int` is authentication plus integrity checking (an MD5 hash of the entity is also computed and checked); `none` will cause the module to use the old RFC-2069 digest algorithm (which does not include integrity checking). Both `auth` and `auth-int` may be specified, in which the case the browser will choose which of these to use. `none` should only be used if the browser for some reason does not like the challenge it receives otherwise.

`auth-int` is not implemented yet.



Description:	The amount of shared memory to allocate for keeping track of clients
Syntax:	<code>AuthDigestShmemSize size</code>
Default:	<code>AuthDigestShmemSize 1000</code>
Context:	server config
Status:	Extension
Module:	<code>mod_auth_digest</code>

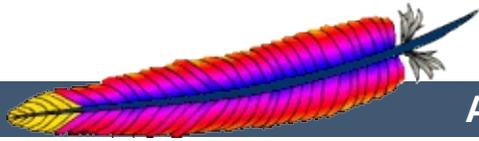
The `AuthDigestShmemSize` directive defines the amount of shared memory, that will be allocated at the server startup for keeping track of clients. Note that the shared memory segment cannot be set less than the space that is necessary for tracking at least *one* client. This value is dependant on your system. If you want to find out the exact value, you may simply set `AuthDigestShmemSize` to the value of 0 and read the error message after trying to start the server.

The *size* is normally expressed in Bytes, but you may let the number follow a K or an M to express your value as KBytes or MBytes. For example, the following directives are all equivalent:

```
AuthDigestShmemSize 1048576
AuthDigestShmemSize 1024K
AuthDigestShmemSize 1M
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_auth_alias`

Description:	Provides the ability to create extended authentication providers based on actual providers
Status:	Extension
Module Identifier:	<code>auth_alias_module</code>
Source File:	<code>mod_auth_alias.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module allows extended authentication providers to be created within the configuration file and assigned an alias name. The alias providers can then be referenced through the directives [AuthBasicProvider](#) or [AuthDigestProvider](#) in the same way as a base authentication provider. Besides the ability to create and alias an extended provider, it also allows the same extended authentication provider to be reference by multiple locations.



This example checks for passwords in two different text files.

Checking multiple text password files

```
# Check here first
<AuthnProviderAlias file file1>
  AuthUserFile /www/conf/passwords1
</AuthnProviderAlias>

# Then check here
<AuthnProviderAlias file file2>
  AuthUserFile /www/conf/passwords2
</AuthnProviderAlias>

<Directory /var/web/pages/secure>
  AuthBasicProvider file1 file2

  AuthType Basic
  AuthName "Protected Area"
  Require valid-user
</Directory>
```

The example below creates two different ldap authentication provider aliases based on the ldap provider. This allows a single authenticated location to be serviced by multiple ldap hosts:

Checking multiple LDAP servers

```
LoadModule authn_alias_module modules/mod_authn_alias.so

<AuthnProviderAlias ldap ldap-alias1>
  AuthLDAPBindDN cn=youruser,o=ctx
  AuthLDAPBindPassword yourpassword
  AuthLDAPURL ldap://ldap.host/o=ctx
</AuthnProviderAlias>

<AuthnProviderAlias ldap ldap-other-alias>
  AuthLDAPBindDN cn=yourotheruser,o=dev
  AuthLDAPBindPassword yourotherpassword
  AuthLDAPURL ldap://other.ldap.host/o=dev?cn
</AuthnProviderAlias>

Alias /secure /webpages/secure
<Directory /webpages/secure>
  Order deny,allow
```

```
Allow from all

AuthBasicProvider ldap-other-alias ldap-alias1

AuthType Basic
AuthName LDAP_Protected_Place
AuthzLDAPAuthoritative off
Require valid-user
</Directory>
```



Description: Enclose a group of directives that represent an extension of a base authentication provider and referenced by the specified alias

Syntax: `<AuthnProviderAlias baseProvider Alias>`
`... </AuthnProviderAlias>`

Context: server config

Status: Extension

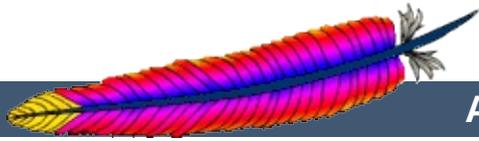
Module: mod_auth_alias

`<AuthnProviderAlias>` and `</AuthnProviderAlias>` are used to enclose a group of authentication directives that can be referenced by the alias name using one of the directives [AuthBasicProvider](#) or [AuthDigestProvider](#).

This directive has no affect on authorization, even for modules that provide both authentication and authorization.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_auth_anon`

Description:	Allows "anonymous" user access to authenticated areas
Status:	Extension
Module Identifier:	<code>authn_anon_module</code>
Source File:	<code>mod_auth_anon.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module provides authentication front-ends such as `mod_auth_basic` to authenticate users similar to anonymous-ftp sites, *i.e.* have a 'magic' user id 'anonymous' and the email address as a password. These email addresses can be logged.

Combined with other (database) access control methods, this allows for effective user tracking and customization according to a user profile while still keeping the site open for 'unregistered' users. One advantage of using Auth-based user tracking is that, unlike magic-cookies and funny URL pre/postfixes, it is completely browser independent and it allows users to share URLs.

When using `mod_auth_basic`, this module is invoked via the `AuthBasicProvider` directive with the `anon` value.



The example below is combined with "normal" htpasswd-file based authentication and allows users in additionally as 'guests' with the following properties:

- It insists that the user enters a userID. ([Anonymous_NoUserID](#))
- It insists that the user enters a password. ([Anonymous_MustGiveEmail](#))
- The password entered must be a valid email address, *i.e.* contain at least one '@' and a '.'. ([Anonymous_VerifyEmail](#))
- The userID must be one of anonymous guest www test welcome and comparison is **not** case sensitive. ([Anonymous](#))
- And the Email addresses entered in the passwd field are logged to the error log file. ([Anonymous_LogEmail](#))

Example

```
<Directory /foo>
  AuthName "Use 'anonymous' & Email address for guest entry"
  AuthType Basic
  AuthBasicProvider file anon
  AuthUserFile /path/to/your/.htpasswd

  Anonymous_NoUserID off
  Anonymous_MustGiveEmail on
  Anonymous_VerifyEmail on
  Anonymous_LogEmail on
  Anonymous anonymous guest www test welcome

  Order Deny,Allow
  Allow from all

  Require valid-user
</Directory>
```



Description: Specifies userIDs that are allowed access without password verification

Syntax: `Anonymous user [user] ...`

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_authn_anon

A list of one or more 'magic' userIDs which are allowed access without password verification. The userIDs are space separated. It is possible to use the ' and " quotes to allow a space in a userID as well as the \ escape character.

Please note that the comparison is **case-IN-sensitive**.

It's strongly recommended that the magic username 'anonymous' is always one of the allowed userIDs.

Example:

```
Anonymous anonymous "Not Registered" "I don't know"
```

This would allow the user to enter without password verification by using the userIDs "anonymous", "AnonyMous", "Not Registered" and "I Don't Know".

As of Apache 2.1 it is possible to specify the userID as "*". That allows any supplied userID to be accepted.



Description:	Sets whether the password entered will be logged in the error log
Syntax:	Anonymous_LogEmail On Off
Default:	Anonymous_LogEmail On
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authn_anon

When set On, the default, the 'password' entered (which hopefully contains a sensible email address) is logged in the error log.



Description:	Specifies whether blank passwords are allowed
Syntax:	Anonymous_MustGiveEmail On Off
Default:	Anonymous_MustGiveEmail On
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authn_anon

Specifies whether the user must specify an email address as the password. This prohibits blank passwords.



Description:	Sets whether the userID field may be empty
Syntax:	Anonymous_NoUserID On Off
Default:	Anonymous_NoUserID Off
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authn_anon

When set On, users can leave the userID (and perhaps the password field) empty. This can be very convenient for MS-Explorer users who can just hit return or click directly on the OK button; which seems a natural reaction.

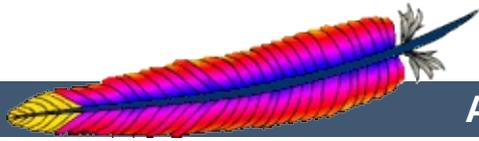


Description:	Sets whether to check the password field for a correctly formatted email address
Syntax:	Anonymous_VerifyEmail On Off
Default:	Anonymous_VerifyEmail Off
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authn_anon

When set On the 'password' entered is checked for at least one '@' and a '.' to encourage users to enter valid email addresses (see the above [Anonymous_LogEmail](#)).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authn_dbd`

Description:	User authentication using an SQL database
Status:	Extension
Module Identifier:	<code>authn_dbd_module</code>
Source File:	<code>mod_authn_dbd.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module provides authentication front-ends such as [mod_auth_digest](#) and [mod_auth_basic](#) to authenticate users by looking up users in SQL tables. Similar functionality is provided by, for example, [mod_authn_file](#).

This module relies on [mod_dbd](#) to specify the backend database driver and connection parameters, and manage the database connections.

When using [mod_auth_basic](#) or [mod_auth_digest](#), this module is invoked via the [AuthBasicProvider](#) or [AuthDigestProvider](#) with the `dbd` value.

See also

- [AuthName](#)
- [AuthType](#)
- [AuthBasicProvider](#)
- [AuthDigestProvider](#)
- [DBDriver](#)
- [DBDParams](#)



Configuration Example

This simple example shows use of this module in the context of the Authentication and DBD frameworks. Please note that you need to load an authorization module, such as [mod_authz_user](#), to get it working.

```
# mod_dbd configuration
DBDriver pgsql
DBDParams "dbname=apacheauth user=apache password=xxxxxxx"

DBDMin 4
DBDKeep 8
DBDMax 20
DBDExptime 300

<Directory /usr/www/myhost/private>
  # core authentication and mod_auth_basic configuration
  # for mod_authn_dbd
  AuthType Basic
  AuthName "My Server"
  AuthBasicProvider dbd

  # core authorization configuration
  Require valid-user

  # mod_authn_dbd SQL query to authenticate a user
  AuthDBDUserPWQuery \
    "SELECT password FROM authn WHERE user = %s"
</Directory>
```



Exposing Login Information

If httpd was built against [APR](#) version 1.3.0 or higher, then whenever a query is made to the database server, all column values in the first row returned by the query are placed in the environment, using environment variables with the prefix "AUTHENTICATE_".

If a database query for example returned the username, full name and telephone number of a user, a CGI program will have access to this information without the need to make a second independent database query to gather this additional information.

This has the potential to dramatically simplify the coding and configuration required in some web applications.



Description:	SQL query to look up a password for a user
Syntax:	<code>AuthDBDUserPWQuery query</code>
Context:	directory
Status:	Extension
Module:	<code>mod_authn_dbd</code>

The `AuthDBDUserPWQuery` specifies an SQL query to look up a password for a specified user. The user's ID will be passed as a single string parameter when the SQL query is executed. It may be referenced within the query statement using a `%s` format specifier.

Example

```
AuthDBDUserPWQuery \  
"SELECT password FROM authn WHERE user = %s"
```

The first column value of the first row returned by the query statement should be a string containing the encrypted password. Subsequent rows will be ignored. If no rows are returned, the user will not be authenticated through `mod_authn_dbd`.

If `httpd` was built against `APR` version 1.3.0 or higher, any additional column values in the first row returned by the query statement will be stored as environment variables with names of the form `AUTHENTICATE_COLUMN`.

The encrypted password format depends on which authentication frontend (e.g. `mod_auth_basic` or `mod_auth_digest`) is being used. See [Password Formats](#) for more information.



Description:	SQL query to look up a password hash for a user and realm.
Syntax:	<code>AuthDBDUserRealmQuery <i>query</i></code>
Context:	directory
Status:	Extension
Module:	<code>mod_authn_dbd</code>

The `AuthDBDUserRealmQuery` specifies an SQL query to look up a password for a specified user and realm in a digest authentication process. The user's ID and the realm, in that order, will be passed as string parameters when the SQL query is executed. They may be referenced within the query statement using `%s` format specifiers.

Example

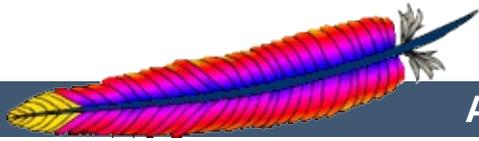
```
AuthDBDUserRealmQuery \  
"SELECT password FROM authn WHERE user = %s AND realm = %s"
```

The first column value of the first row returned by the query statement should be a string containing the encrypted password. Subsequent rows will be ignored. If no rows are returned, the user will not be authenticated through `mod_authn_dbd`.

If `httpd` was built against `APR` version 1.3.0 or higher, any additional column values in the first row returned by the query statement will be stored as environment variables with names of the form `AUTHENTICATE_COLUMN`.

The encrypted password format depends on which authentication frontend (e.g. `mod_auth_basic` or `mod_auth_digest`) is being used. See [Password Formats](#) for more information.

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_auth_dbm`

Description:	User authentication using DBM files
Status:	Extension
Module Identifier:	<code>authn_dbm_module</code>
Source File:	<code>mod_auth_dbm.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module provides authentication front-ends such as [mod_auth_digest](#) and [mod_auth_basic](#) to authenticate users by looking up users in *dbm* password files. Similar functionality is provided by [mod_authn_file](#).

When using [mod_auth_basic](#) or [mod_auth_digest](#), this module is invoked via the [AuthBasicProvider](#) or [AuthDigestProvider](#) with the `dbm` value.

See also

[AuthName](#)

[AuthType](#)

[AuthBasicProvider](#)

[AuthDigestProvider](#)



Description:	Sets the type of database file that is used to store passwords
Syntax:	AuthDBMType default SDBM GDBM NDBM DB
Default:	AuthDBMType default
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authn_dbm

Sets the type of database file that is used to store the passwords. The default database type is determined at compile time. The availability of other types of database files also depends on [compile-time settings](#).

It is crucial that whatever program you use to create your password files is configured to use the same type of database.



Description: Sets the name of a database file containing the list of users and passwords for authentication

Syntax: AuthDBMUserFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_authn_dbm

The `AuthDBMUserFile` directive sets the name of a DBM file containing the list of users and passwords for user authentication. *File-path* is the absolute path to the user file.

The user file is keyed on the username. The value for a user is the encrypted password, optionally followed by a colon and arbitrary data. The colon and the data following it will be ignored by the server.

Security:

Make sure that the `AuthDBMUserFile` is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the `AuthDBMUserFile`.

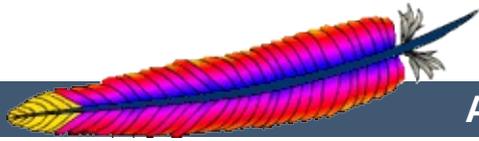
Important compatibility note: The implementation of `dbmopen` in the apache modules reads the string length of the hashed values from the DBM data structures, rather than relying upon the string being NULL-appended. Some applications, such as the Netscape web server, rely upon the string being NULL-appended, so if you are having trouble using DBM files interchangeably between applications this may be a part of the problem.

A perl script called `dbmmanage` is included with Apache. This program can be used to create and update DBM format password files for use with

this module.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_auth_default`

Description:	Authentication fallback module
Status:	Base
Module Identifier:	<code>authn_default_module</code>
Source File:	<code>mod_auth_default.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module is designed to be the fallback module, if you don't have configured an authentication module like `mod_auth_basic`. It simply rejects any credentials supplied by the user.



Description:	Sets whether authentication is passed to lower level modules
Syntax:	<code>AuthDefaultAuthoritative On Off</code>
Default:	<code>AuthDefaultAuthoritative On</code>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	<code>mod_auth_default</code>

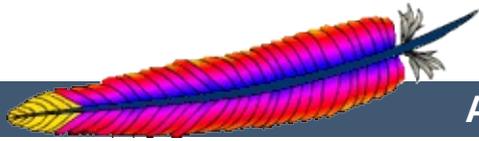
Setting the `AuthDefaultAuthoritative` directive explicitly to `Off` allows for authentication to be passed on to lower level modules (as defined in the `modules.c` files).

Note

Normally there are no lower level modules, since `mod_auth_default` is defined to be already on a *very low* level. Therefore you should leave the value of `AuthDefaultAuthoritative` as default (`On`).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_auth_file`

Description:	User authentication using text files
Status:	Base
Module Identifier:	<code>auth_file_module</code>
Source File:	<code>mod_auth_file.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module provides authentication front-ends such as [mod_auth_digest](#) and [mod_auth_basic](#) to authenticate users by looking up users in plain text password files. Similar functionality is provided by [mod_authn_dbm](#).

When using [mod_auth_basic](#) or [mod_auth_digest](#), this module is invoked via the [AuthBasicProvider](#) or [AuthDigestProvider](#) with the `file` value.

See also

[AuthBasicProvider](#)
[AuthDigestProvider](#)
[htpasswd](#)
[htdigest](#)



Description:	Sets the name of a text file containing the list of users and passwords for authentication
Syntax:	<code>AuthUserFile file-path</code>
Context:	directory, <code>.htaccess</code>
Override:	<code>AuthConfig</code>
Status:	Base
Module:	<code>mod_authn_file</code>

The `AuthUserFile` directive sets the name of a textual file containing the list of users and passwords for user authentication. *File-path* is the path to the user file. If it is not absolute, it is treated as relative to the `ServerRoot`.

Each line of the user file contains a username followed by a colon, followed by the encrypted password. If the same user ID is defined multiple times, `mod_authn_file` will use the first occurrence to verify the password.

The utility `htpasswd` which is installed as part of the binary distribution, or which can be found in `src/support`, is used to maintain the password file for *HTTP Basic Authentication*. See the [man page](#) for more details. In short:

Create a password file `Filename` with `username` as the initial ID. It will prompt for the password:

```
htpasswd -c Filename username
```

Add or modify `username2` in the password file `Filename`:

```
htpasswd Filename username2
```

Note that searching large text files is very inefficient; [AuthDBMUserFile](#)

should be used instead.

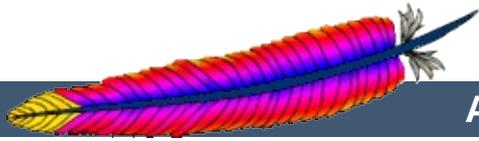
If you are using *HTTP Digest Authentication*, the [htpasswd](#) tool is not sufficient. You have to use [htdigest](#) instead. Note that you cannot mix user data for Digest Authentication and Basic Authentication within the same file.

Security

Make sure that the `AuthUserFile` is stored outside the document tree of the web-server. Do **not** put it in the directory that it protects. Otherwise, clients may be able to download the `AuthUserFile`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authnz_ldap`

Description:	Allows an LDAP directory to be used to store the database for HTTP Basic authentication.
Status:	Extension
Module Identifier:	<code>authnz_ldap_module</code>
Source File:	<code>mod_authnz_ldap.c</code>
Compatibility:	Available in version 2.1 and later

Summary

This module provides authentication front-ends such as `mod_auth_basic` to authenticate users through an ldap directory.

`mod_authnz_ldap` supports the following features:

- Known to support the [OpenLDAP SDK](#) (both 1.x and 2.x), [Novell LDAP SDK](#) and the [iPlanet \(Netscape\) SDK](#).
- Complex authorization policies can be implemented by representing the policy with LDAP filters.
- Uses extensive caching of LDAP operations via [mod_ldap](#).
- Support for LDAP over SSL (requires the Netscape SDK) or TLS (requires the OpenLDAP 2.x SDK or Novell LDAP SDK).

When using `mod_auth_basic`, this module is invoked via the [AuthBasicProvider](#) directive with the `ldap` value.

See also

[mod_ldap](#)
[mod_auth_basic](#)
[mod_authz_user](#)
[mod_authz_groupfile](#)



-
- [Operation](#)
 - [The Authentication Phase](#)
 - [The Authorization Phase](#)

 - [The Require Directives](#)
 - [Require ldap-user](#)
 - [Require ldap-group](#)
 - [Require ldap-dn](#)
 - [Require ldap-attribute](#)
 - [Require ldap-filter](#)

 - [Examples](#)
 - [Using TLS](#)
 - [Using SSL](#)
 - [Exposing Login Information](#)
 - [Using Microsoft FrontPage with mod_authnz_ldap](#)
 - [How It Works](#)
 - [Caveats](#)



There are two phases in granting access to a user. The first phase is authentication, in which the `mod_authnz_ldap` authentication provider verifies that the user's credentials are valid. This is also called the *search/bind* phase. The second phase is authorization, in which `mod_authnz_ldap` determines if the authenticated user is allowed access to the resource in question. This is also known as the *compare* phase.

`mod_authnz_ldap` registers both an `authn_ldap` authentication provider and an `authz_ldap` authorization handler. The `authn_ldap` authentication provider can be enabled through the `AuthBasicProvider` directive using the `ldap` value. The `authz_ldap` handler extends the `Require` directive's authorization types by adding `ldap-user`, `ldap-dn` and `ldap-group` values.

The Authentication Phase

During the authentication phase, `mod_authnz_ldap` searches for an entry in the directory that matches the username that the HTTP client passes. If a single unique match is found, then `mod_authnz_ldap` attempts to bind to the directory server using the DN of the entry plus the password provided by the HTTP client. Because it does a search, then a bind, it is often referred to as the *search/bind* phase. Here are the steps taken during the *search/bind* phase.

1. Generate a search filter by combining the attribute and filter provided in the `AuthLDAPURL` directive with the username passed by the HTTP client.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny or decline access.
3. Fetch the distinguished name of the entry retrieved from the search and attempt to bind to the LDAP server using the DN and the password passed by the HTTP client. If the bind is unsuccessful,

deny or decline access.

The following directives are used during the search/bind phase

<u>AuthLDAPURL</u>	Specifies the LDAP server, the base DN, the attribute to use in the search, as well as the extra search filter to use.
<u>AuthLDAPBindDN</u>	An optional DN to bind with during the search phase.
<u>AuthLDAPBindPassword</u>	An optional password to bind with during the search phase.

The Authorization Phase

During the authorization phase, [mod_authnz_ldap](#) attempts to determine if the user is authorized to access the resource. Many of these checks require [mod_authnz_ldap](#) to do a compare operation on the LDAP server. This is why this phase is often referred to as the compare phase. [mod_authnz_ldap](#) accepts the following [Require](#) directives to determine if the credentials are acceptable:

- Grant access if there is a [Require ldap-user](#) directive, and the username in the directive matches the username passed by the client.
- Grant access if there is a [Require ldap-dn](#) directive, and the DN in the directive matches the DN fetched from the LDAP directory.
- Grant access if there is a [Require ldap-group](#) directive, and the DN fetched from the LDAP directory (or the username passed by the client) occurs in the LDAP group.
- Grant access if there is a [Require ldap-attribute](#) directive, and the attribute fetched from the LDAP directory matches the given value.
- Grant access if there is a [Require ldap-filter](#) directive, and the search filter successfully finds a single user object that matches

- the dn of the authenticated user.
- otherwise, deny or decline access

Other [Require](#) values may also be used which may require loading additional authorization modules. Note that if you use a [Require](#) value from another authorization module, you will need to ensure that [AuthzLDAPAuthoritative](#) is set to off to allow the authorization phase to fall back to the module providing the alternate [Require](#) value. When no LDAP-specific [Require](#) directives are used, authorization is allowed to fall back to other modules as if [AuthzLDAPAuthoritative](#) was set to off.

- Grant access to all successfully authenticated users if there is a [Require valid-user](#) directive. (requires [mod_authz_user](#))
- Grant access if there is a [Require group](#) directive, and [mod_authz_groupfile](#) has been loaded with the [AuthGroupFile](#) directive set.
- others...

[mod_authnz_ldap](#) uses the following directives during the compare phase:

AuthLDAPURL	The attribute specified in the URL is used in compare operations for the Require ldap-user operation.
AuthLDAPCompareDNOnServer	Determines the behavior of the Require ldap-dn directive.
AuthLDAPGroupAttribute	Determines the attribute to use for comparisons in the Require ldap-group directive.
AuthLDAPGroupAttributeIsDN	Specifies whether to use the user DN or the username when doing comparisons for the Require

ldap-group directive.



Apache's [Require](#) directives are used during the authorization phase to ensure that a user is allowed to access a resource. `mod_authnz_ldap` extends the authorization types with `ldap-user`, `ldap-dn`, `ldap-group`, `ldap-attribute` and `ldap-filter`. Other authorization types may also be used but may require that additional authorization modules be loaded.

Require ldap-user

The `Require ldap-user` directive specifies what usernames can access the resource. Once `mod_authnz_ldap` has retrieved a unique DN from the directory, it does an LDAP compare operation using the username specified in the `Require ldap-user` to see if that username is part of the just-fetched LDAP entry. Multiple users can be granted access by putting multiple usernames on the line, separated with spaces. If a username has a space in it, then it must be surrounded with double quotes. Multiple users can also be granted access by using multiple `Require ldap-user` directives, with one user per line. For example, with a [AuthLDAPURL](#) of `ldap://ldap/o=Airius?cn` (i.e., `cn` is used for searches), the following `Require` directives could be used to restrict access:

```
Require ldap-user "Barbara Jenson"  
Require ldap-user "Fred User"  
Require ldap-user "Joe Manager"
```

Because of the way that `mod_authnz_ldap` handles this directive, Barbara Jenson could sign on as *Barbara Jenson*, *Babs Jenson* or any other `cn` that she has in her LDAP entry. Only the single `Require ldap-user` line is needed to support all values of the attribute in the user's entry.

If the `uid` attribute was used instead of the `cn` attribute in the URL above, the above three lines could be condensed to

```
Require ldap-user bjenson fuser jmanager
```

Require ldap-group

This directive specifies an LDAP group whose members are allowed access. It takes the distinguished name of the LDAP group. Note: Do not surround the group name with quotes. For example, assume that the following entry existed in the LDAP directory:

```
dn: cn=Administrators, o=Airius
objectClass: groupOfUniqueNames
uniqueMember: cn=Barbara Jenson, o=Airius
uniqueMember: cn=Fred User, o=Airius
```

The following directive would grant access to both Fred and Barbara:

```
Require ldap-group cn=Administrators, o=Airius
```

Behavior of this directive is modified by the [AuthLDAPGroupAttribute](#) and [AuthLDAPGroupAttributeIsDN](#) directives.

Require ldap-dn

The `Require ldap-dn` directive allows the administrator to grant access based on distinguished names. It specifies a DN that must match for access to be granted. If the distinguished name that was retrieved from the directory server matches the distinguished name in the `Require ldap-dn`, then authorization is granted. Note: do not surround the distinguished name with quotes.

The following directive would grant access to a specific DN:

```
Require ldap-dn cn=Barbara Jenson, o=Airius
```

Behavior of this directive is modified by the

[AuthLDAPCompareDNOnServer](#) directive.

Require ldap-attribute

The `Require ldap-attribute` directive allows the administrator to grant access based on attributes of the authenticated user in the LDAP directory. If the attribute in the directory matches the value given in the configuration, access is granted.

The following directive would grant access to anyone with the attribute `employeeType = active`

```
Require ldap-attribute employeeType=active
```

Multiple attribute/value pairs can be specified on the same line separated by spaces or they can be specified in multiple `Require ldap-attribute` directives. The effect of listing multiple attribute/values pairs is an OR operation. Access will be granted if any of the listed attribute values match the value of the corresponding attribute in the user object. If the value of the attribute contains a space, only the value must be within double quotes.

The following directive would grant access to anyone with the city attribute equal to "San Jose" or status equal to "Active"

```
Require ldap-attribute city="San Jose" status=active
```

Require ldap-filter

The `Require ldap-filter` directive allows the administrator to grant access based on a complex LDAP search filter. If the dn returned by the filter search matches the authenticated user dn, access is granted.

The following directive would grant access to anyone having a cell phone and is in the marketing department

```
Require ldap-filter &(cell=*)(department=marketing)
```

The difference between the `Require ldap-filter` directive and the `Require ldap-attribute` directive is that `ldap-filter` performs a search operation on the LDAP directory using the specified search filter rather than a simple attribute comparison. If a simple attribute comparison is all that is required, the comparison operation performed by `ldap-attribute` will be faster than the search operation used by `ldap-filter` especially within a large directory.



- Grant access to anyone who exists in the LDAP directory, using their UID for searches.

```
AuthLDAPURL "ldap://ldap1.airius.com:389/ou=People, o=Airius?uid?sub?(objectClass=*)"
Require valid-user
```

- The next example is the same as above; but with the fields that have useful defaults omitted. Also, note the use of a redundant LDAP server.

```
AuthLDAPURL "ldap://ldap1.airius.com ldap2.airius.com/ou=People, o=Airius"
Require valid-user
```

- The next example is similar to the previous one, but it uses the common name instead of the UID. Note that this could be problematical if multiple people in the directory share the same cn, because a search on cn **must** return exactly one entry. That's why this approach is not recommended: it's a better idea to choose an attribute that is guaranteed unique in your directory, such as uid.

```
AuthLDAPURL "ldap://ldap.airius.com/ou=People, o=Airius?cn"
Require valid-user
```

- Grant access to anybody in the Administrators group. The users must authenticate using their UID.

```
AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid
Require ldap-group cn=Administrators, o=Airius
```

- The next example assumes that everyone at Airius who carries an alphanumeric pager will have an LDAP attribute of qpagePager ID. The example will grant access only to people (authenticated via their

UID) who have alphanumeric pagers:

```
AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid??(qpagePagerID=*)
Require valid-user
```

- The next example demonstrates the power of using filters to accomplish complicated administrative requirements. Without filters, it would have been necessary to create a new LDAP group and ensure that the group's members remain synchronized with the pager users. This becomes trivial with filters. The goal is to grant access to anyone who has a pager, plus grant access to Joe Manager, who doesn't have a pager, but does need to access the same resource:

```
AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid??(|
(qpagePagerID=*)(uid=jmanager))
Require valid-user
```

This last may look confusing at first, so it helps to evaluate what the search filter will look like based on who connects, as shown below. If Fred User connects as *fuser*, the filter would look like

```
(&( |(qpagePagerID=*)(uid=jmanager))(uid=fuser))
```

The above search will only succeed if *fuser* has a pager. When Joe Manager connects as *jmanager*, the filter looks like

```
(&( |(qpagePagerID=*)(uid=jmanager))(uid=jmanager))
```

The above search will succeed whether *jmanager* has a pager or not.



To use TLS, see the `mod_ldap` directives [LDAPTrustedClientCert](#), [LDAPTrustedGlobalCert](#) and [LDAPTrustedMode](#).

An optional second parameter can be added to the [AuthLDAPURL](#) to override the default connection type set by [LDAPTrustedMode](#). This will allow the connection established by an `ldap://` Url to be upgraded to a secure connection on the same port.



To use SSL, see the `mod_ldap` directives [LDAPTrustedClientCert](#), [LDAPTrustedGlobalCert](#) and [LDAPTrustedMode](#).

To specify a secure LDAP server, use `ldaps://` in the [AuthLDAPURL](#) directive, instead of `ldap://`.



Exposing Login Information

When this module performs authentication, LDAP attributes specified in the `AuthLDAPUrl` directive are placed in environment variables with the prefix "AUTHENTICATE_".

If the attribute field contains the username, common name and telephone number of a user, a CGI program will have access to this information without the need to make a second independent LDAP query to gather this additional information.

This has the potential to dramatically simplify the coding and configuration required in some web applications.



Normally, FrontPage uses FrontPage-web-specific user/group files (i.e., the `mod_authn_file` and `mod_authz_groupfile` modules) to handle all authentication. Unfortunately, it is not possible to just change to LDAP authentication by adding the proper directives, because it will break the *Permissions* forms in the FrontPage client, which attempt to modify the standard text-based authorization files.

Once a FrontPage web has been created, adding LDAP authentication to it is a matter of adding the following directives to every `.htaccess` file that gets created in the web

```
AuthLDAPURL          "the url"  
AuthGroupFile mygroupfile  
Require group mygroupfile
```

How It Works

FrontPage restricts access to a web by adding the `Require valid-user` directive to the `.htaccess` files. The `Require valid-user` directive will succeed for any user who is valid *as far as LDAP is concerned*. This means that anybody who has an entry in the LDAP directory is considered a valid user, whereas FrontPage considers only those people in the local user file to be valid. By substituting the `ldap-group` with `group` file authorization, Apache is allowed to consult the local user file (which is managed by FrontPage) - instead of LDAP - when handling authorizing the user.

Once directives have been added as specified above, FrontPage users will be able to perform all management operations from the FrontPage client.

Caveats

- When choosing the LDAP URL, the attribute to use for authentication should be something that will also be valid for putting into a

[mod_authn_file](#) user file. The user ID is ideal for this.

- When adding users via FrontPage, FrontPage administrators should choose usernames that already exist in the LDAP directory (for obvious reasons). Also, the password that the administrator enters into the form is ignored, since Apache will actually be authenticating against the password in the LDAP database, and not against the password in the local user file. This could cause confusion for web administrators.
- Apache must be compiled with [mod_auth_basic](#), [mod_authn_file](#) and [mod_authz_groupfile](#) in order to use FrontPage support. This is because Apache will still use the [mod_authz_groupfile](#) group file for determine the extent of a user's access to the FrontPage web.
- The directives must be put in the .htaccess files. Attempting to put them inside [<Location>](#) or [<Directory>](#) directives won't work. This is because [mod_authnz_ldap](#) has to be able to grab the [AuthGroupFile](#) directive that is found in FrontPage .htaccess files so that it knows where to look for the valid user list. If the [mod_authnz_ldap](#) directives aren't in the same .htaccess file as the FrontPage directives, then the hack won't work, because [mod_authnz_ldap](#) will never get a chance to process the .htaccess file, and won't be able to find the FrontPage-managed user file.



Description:	Determines if other authentication providers are used when a user can be mapped to a DN but the server cannot successfully bind with the user's credentials.
Syntax:	<code>AuthLDAPBindAuthoritativeoff on</code>
Default:	<code>AuthLDAPBindAuthoritative on</code>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	<code>mod_authnz_ldap</code>
Compatibility:	Available in versions later than 2.2.14

By default, subsequent authentication providers are only queried if a user cannot be mapped to a DN, but not if the user can be mapped to a DN and their password cannot be verified with an LDAP bind. If `AuthLDAPBindAuthoritative` is set to `off`, other configured authentication modules will have a chance to validate the user if the LDAP bind (with the current user's credentials) fails for any reason.

This allows users present in both LDAP and `AuthUserFile` to authenticate when the LDAP server is available but the user's account is locked or password is otherwise unusable.

See also

- [AuthUserFile](#)
- [AuthBasicProvider](#)



Description:	Optional DN to use in binding to the LDAP server
Syntax:	AuthLDAPBindDN <i>distinguished-name</i>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

An optional DN used to bind to the server when searching for entries. If not provided, [mod_authnz_ldap](#) will use an anonymous bind.



Description:	Password used in conjunction with the bind DN
Syntax:	AuthLDAPBindPassword <i>password</i>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap
Compatibility:	exec: was added in 2.2.25.

A bind password to use in conjunction with the bind DN. Note that the bind password is probably sensitive data, and should be properly protected. You should only use the [AuthLDAPBindDN](#) and [AuthLDAPBindPassword](#) if you absolutely need them to search the directory.

If the value begins with exec: the resulting command will be executed and the first line returned to standard output by the program will be used as the password.

```
#Password used as-is
AuthLDAPBindPassword secret

#Run /path/to/program to get my password
AuthLDAPBindPassword exec:/path/to/program

#Run /path/to/otherProgram and provide arguments
AuthLDAPBindPassword "exec:/path/to/otherProgram argument1"
```



Description:	Language to charset conversion configuration file
Syntax:	<code>AuthLDAPCharsetConfig</code> <i>file-path</i>
Context:	server config
Status:	Extension
Module:	<code>mod_authnz_ldap</code>

The `AuthLDAPCharsetConfig` directive sets the location of the language to charset conversion configuration file. *File-path* is relative to the `ServerRoot`. This file specifies the list of language extensions to character sets. Most administrators use the provided `charset.conv` file, which associates common language extensions to character sets.

The file contains lines in the following format:

```
Language-Extension charset [Language-String] ...
```

The case of the extension does not matter. Blank lines, and lines beginning with a hash character (#) are ignored.



Description:	Use the LDAP server to compare the DN's
Syntax:	AuthLDAPCompareDNOnServer on off
Default:	AuthLDAPCompareDNOnServer on
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

When set, [mod_authnz_ldap](#) will use the LDAP server to compare the DN's. This is the only foolproof way to compare DN's. [mod_authnz_ldap](#) will search the directory for the DN specified with the [Require dn](#) directive, then, retrieve the DN and compare it with the DN retrieved from the user entry. If this directive is not set, [mod_authnz_ldap](#) simply does a string comparison. It is possible to get false negatives with this approach, but it is much faster. Note the [mod_ldap](#) cache can speed up DN comparison in most situations.



Description:	When will the module de-reference aliases
Syntax:	AuthLDAPDereferenceAliases never searching finding always
Default:	AuthLDAPDereferenceAliases Always
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

This directive specifies when `mod_authnz_ldap` will de-reference aliases during LDAP operations. The default is `always`.



Description:	LDAP attributes used to check for group membership
Syntax:	AuthLDAPGroupAttribute <i>attribute</i>
Default:	AuthLDAPGroupAttribute member uniquemember
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

This directive specifies which LDAP attributes are used to check for group membership. Multiple attributes can be used by specifying this directive multiple times. If not specified, then [mod_authnz_ldap](#) uses the member and uniquemember attributes.



Description:	Use the DN of the client username when checking for group membership
Syntax:	AuthLDAPGroupAttributeIsDN on off
Default:	AuthLDAPGroupAttributeIsDN on
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

When set on, this directive says to use the distinguished name of the client username when checking for group membership. Otherwise, the username will be used. For example, assume that the client sent the username bjenson, which corresponds to the LDAP DN cn=Babs Jenson, o=Airius. If this directive is set, [mod_authnz_ldap](#) will check if the group has cn=Babs Jenson, o=Airius as a member. If this directive is not set, then [mod_authnz_ldap](#) will check if the group has bjenson as a member.



Description:	Use the value of the attribute returned during the user query to set the REMOTE_USER environment variable
Syntax:	AuthLDAPRemoteUserAttribute uid
Default:	none
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

If this directive is set, the value of the REMOTE_USER environment variable will be set to the value of the attribute specified. Make sure that this attribute is included in the list of attributes in the AuthLDAPUrl definition, otherwise this directive will have no effect. This directive, if present, takes precedence over AuthLDAPRemoteUserIsDN. This directive is useful should you want people to log into a website using an email address, but a backend application expects the username as a userid.



Description:	Use the DN of the client username to set the REMOTE_USER environment variable
Syntax:	AuthLDAPRemoteUserIsDN on off
Default:	AuthLDAPRemoteUserIsDN off
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

If this directive is set to on, the value of the REMOTE_USER environment variable will be set to the full distinguished name of the authenticated user, rather than just the username that was passed by the client. It is turned off by default.



Description:	URL specifying the LDAP search parameters
Syntax:	AuthLDAPUrl <i>url</i> [<i>NONE</i> <i>SSL</i> <i>TLS</i> <i>STARTTLS</i>]
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

An RFC 2255 URL which specifies the LDAP search parameters to use. The syntax of the URL is

```
ldap://host:port/basedn?attribute?scope?filter
```

ldap

For regular ldap, use the string ldap. For secure LDAP, use ldaps instead. Secure LDAP is only available if Apache was linked to an LDAP library with SSL support.

host:port

The name/port of the ldap server (defaults to localhost : 389 for ldap, and localhost : 636 for ldaps). To specify multiple, redundant LDAP servers, just list all servers, separated by spaces. [mod_authnz_ldap](#) will try connecting to each server in turn, until it makes a successful connection.

Once a connection has been made to a server, that connection remains active for the life of the [httpd](#) process, or until the LDAP server goes down.

If the LDAP server goes down and breaks an existing connection, [mod_authnz_ldap](#) will attempt to re-connect, starting with the primary server, and trying each redundant server in turn. Note that this is different than a true round-robin search.

basedn

The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but could also specify a subtree in the directory.

attribute

The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use `uid`. It's a good idea to choose an attribute that will be unique across all entries in the subtree you will be using.

scope

The scope of the search. Can be either `one` or `sub`. Note that a scope of `base` is also supported by RFC 2255, but is not supported by this module. If the scope is not provided, or if `base` scope is specified, the default is to use a scope of `sub`.

filter

A valid LDAP search filter. If not provided, defaults to `(objectClass=*)`, which will search for all objects in the tree. Filters are limited to approximately 8000 characters (the definition of `MAX_STRING_LEN` in the Apache source code). This should be more than sufficient for any application.

When doing searches, the attribute, filter and username passed by the HTTP client are combined to create a search filter that looks like `(&(filter)(attribute=username))`.

For example, consider an URL of

```
ldap://ldap.airius.com/o=Airius?cn?sub?(posixid=*)
```

When a client attempts to connect using a username of `Babs Jenson`, the resulting search filter will be `(&(posixid=*)(cn=Babs Jenson))`.

An optional parameter can be added to allow the LDAP Url to override the connection type. This parameter can be one of the following:

NONE

Establish an unsecure connection on the default LDAP port. This is the same as `ldap://` on port 389.

SSL

Establish a secure connection on the default secure LDAP port. This is the same as `ldaps://`

TLS | STARTTLS

Establish an upgraded secure connection on the default LDAP port. This connection will be initiated on port 389 by default and then upgraded to a secure connection on the same port.

See above for examples of [AuthLDAPURL](#) URLs.

When [AuthLDAPURL](#) is enabled in a particular context, but some other module has performed authentication for the request, the server will try to map the username to a DN during authorization regardless of whether or not LDAP-specific requirements are present. To ignore the failures to map a username to a DN during authorization, set [AuthzLDAPAuthoritative](#) to "off".



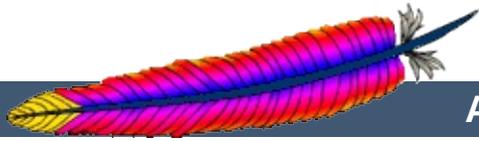
Description:	Prevent other authentication modules from authenticating the user if this one fails
Syntax:	AuthzLDAPAuthoritative on off
Default:	AuthzLDAPAuthoritative on
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authnz_ldap

Set to off if this module should let other authorization modules attempt to authorize the user, should authorization with this module fail. Control is only passed on to lower modules if there is no DN or rule that matches the supplied user name (as passed by the client).

When no LDAP-specific [Require](#) directives are used, authorization is allowed to fall back to other modules as if [AuthzLDAPAuthoritative](#) was set to off.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authz_dbm`

Description:	Group authorization using DBM files
Status:	Extension
Module Identifier:	<code>authz_dbm_module</code>
Source File:	<code>mod_authz_dbm.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the web site by group membership. Similar functionality is provided by [mod_authz_groupfile](#).

See also

[Require](#)
[Satisfy](#)



Description:	Sets the name of the database file containing the list of user groups for authorization
Syntax:	<code>AuthDBMGroupFile <i>file-path</i></code>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authz_dbm

The `AuthDBMGroupFile` directive sets the name of a DBM file containing the list of user groups for user authorization. *File-path* is the absolute path to the group file.

The group file is keyed on the username. The value for a user is a comma-separated list of the groups to which the users belongs. There must be no whitespace within the value, and it must never contain any colons.

Security

Make sure that the `AuthDBMGroupFile` is stored outside the document tree of the web-server. Do **not** put it in the directory that it protects. Otherwise, clients will be able to download the `AuthDBMGroupFile` unless otherwise protected.

Combining Group and Password DBM files: In some cases it is easier to manage a single database which contains both the password and group details for each user. This simplifies any support programs that need to be written: they now only have to deal with writing to and locking a single DBM file. This can be accomplished by first setting the group and password files to point to the same DBM:

```
AuthDBMGroupFile /www/userbase
AuthDBMUserFile /www/userbase
```

The key for the single DBM is the username. The value consists of

```
Encrypted Password : List of Groups [ : (ignored) ]
```

The password section contains the encrypted password as before. This is followed by a colon and the comma separated list of groups. Other data may optionally be left in the DBM file after another colon; it is ignored by the authorization module. This is what www.telescope.org uses for its combined password and group database.



Description:	Sets whether authorization will be passed on to lower level modules
Syntax:	AuthzDBMAuthoritative On Off
Default:	AuthzDBMAuthoritative On
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authz_dbm

Setting the `AuthzDBMAuthoritative` directive explicitly to `Off` allows group authorization to be passed on to lower level modules (as defined in the `modules.c` file) if there is no group found for the supplied `userID`. If there are any groups specified, the usual checks will be applied and a failure will give an Authentication Required reply.

So if a `userID` appears in the database of more than one module; or if a valid `Require` directive applies to more than one module; then the first module will verify the credentials; and no access is passed on; regardless of the `AuthAuthoritative` setting.

A common use for this is in conjunction with one of the auth providers; such as `mod_authn_dbm` or `mod_authn_file`. Whereas this DBM module supplies the bulk of the user credential checking; a few (administrator) related accesses fall through to a lower level with a well protected `.htpasswd` file.

By default, control is not passed on and an unknown group will result in an Authentication Required reply. Not setting it thus keeps the system secure and forces an NCSA compliant behaviour.

Security

Do consider the implications of allowing a user to allow fall-through in

his .htaccess file; and verify that this is really what you want; Generally it is easier to just secure a single .htpasswd file, than it is to secure a database which might have more access interfaces.



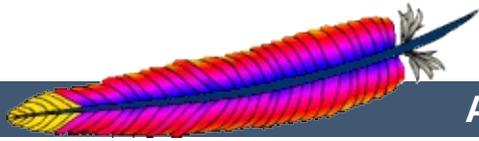
Description:	Sets the type of database file that is used to store list of user groups
Syntax:	AuthzDBMType default SDBM GDBM NDBM DB
Default:	AuthzDBMType default
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_authz_dbm

Sets the type of database file that is used to store the list of user groups. The default database type is determined at compile time. The availability of other types of database files also depends on [compile-time settings](#).

It is crucial that whatever program you use to create your group files is configured to use the same type of database.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authz_default`

Description:	Authorization fallback module
Status:	Base
Module Identifier:	<code>authz_default_module</code>
Source File:	<code>mod_authz_default.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module is designed to be the fallback module, if you don't have configured an authorization module like `mod_authz_user` or `mod_authz_groupfile`. It simply rejects any authorization request.



Description:	Sets whether authorization is passed to lower level modules
Syntax:	<code>AuthzDefaultAuthoritative On Off</code>
Default:	<code>AuthzDefaultAuthoritative On</code>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	<code>mod_authz_default</code>

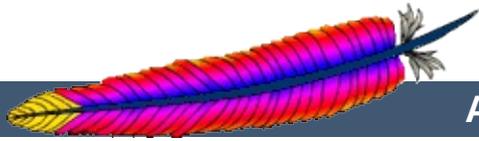
Setting the `AuthzDefaultAuthoritative` directive explicitly to `Off` allows for authorization to be passed on to lower level modules (as defined in the `modules.c` files).

Note

Normally there are no lower level modules, since `mod_authz_default` is defined to be already on a *very low* level. Therefore you should leave the value of `AuthzDefaultAuthoritative` as default (`On`).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authz_groupfile`

Description:	Group authorization using plaintext files
Status:	Base
Module Identifier:	<code>authz_groupfile_module</code>
Source File:	<code>mod_authz_groupfile.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the web site by group membership. Similar functionality is provided by [mod_authz_dbm](#).

See also

[Require](#)
[Satisfy](#)



Description:	Sets the name of a text file containing the list of user groups for authorization
Syntax:	<code>AuthGroupFile file-path</code>
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	mod_authz_groupfile

The `AuthGroupFile` directive sets the name of a textual file containing the list of user groups for user authorization. *File-path* is the path to the group file. If it is not absolute, it is treated as relative to the `ServerRoot`.

Each line of the group file contains a groupname followed by a colon, followed by the member usernames separated by spaces.

Example:

```
mygroup: bob joe anne
```

Note that searching large text files is very inefficient; `AuthDBMGroupFile` provides a much better performance.

Security

Make sure that the `AuthGroupFile` is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients may be able to download the `AuthGroupFile`.



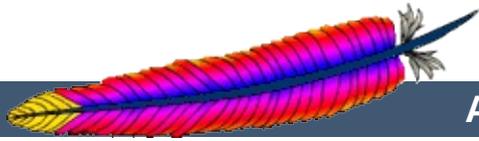
Description:	Sets whether authorization will be passed on to lower level modules
Syntax:	AuthzGroupFileAuthoritative On Off
Default:	AuthzGroupFileAuthoritative On
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	mod_authz_groupfile

Setting the `AuthzGroupFileAuthoritative` directive explicitly to `Off` allows for group authorization to be passed on to lower level modules (as defined in the `modules.c` files) if there is **no group** matching the supplied `userID`.

By default, control is not passed on and an unknown group will result in an Authentication Required reply. Not setting it thus keeps the system secure and forces an NCSA compliant behaviour.

Security

Do consider the implications of allowing a user to allow fall-through in his `.htaccess` file; and verify that this is really what you want; Generally it is easier to just secure a single `.htpasswd` file, than it is to secure a database which might have more access interfaces.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authz_host`

Description:	Group authorizations based on host (name or IP address)
Status:	Base
Module Identifier:	<code>authz_host_module</code>
Source File:	<code>mod_authz_host.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

The directives provided by `mod_authz_host` are used in `<Directory>`, `<Files>`, and `<Location>` sections as well as `.htaccess` files to control access to particular parts of the server. Access can be controlled based on the client hostname, IP address, or other characteristics of the client request, as captured in `environment variables`. The `Allow` and `Deny` directives are used to specify which clients are or are not allowed access to the server, while the `Order` directive sets the default access state, and configures how the `Allow` and `Deny` directives interact with each other.

Both host-based access restrictions and password-based authentication may be implemented simultaneously. In that case, the `Satisfy` directive is used to determine how the two sets of restrictions interact.

In general, access restriction directives apply to all access methods (GET, PUT, POST, etc). This is the desired behavior in most cases. However, it is possible to restrict some methods, while leaving other methods unrestricted, by enclosing the directives in a `<Limit>` section.

See also

[Satisfy](#)

[Require](#)



Description:	Controls which hosts can access an area of the server
Syntax:	Allow from all <i>host</i> env=[!] <i>env-variable</i> [<i>host</i> env=[!] <i>env-variable</i>] ...
Context:	directory, .htaccess
Override:	Limit
Status:	Base
Module:	mod_authz_host
Compatibility:	Support for negating <i>env=env-variable</i> was added in 2.2.10

The **Allow** directive affects which hosts can access an area of the server. Access can be controlled by hostname, IP address, IP address range, or by other characteristics of the client request captured in environment variables.

The first argument to this directive is always `from`. The subsequent arguments can take three different forms. If `Allow from all` is specified, then all hosts are allowed access, subject to the configuration of the **Deny** and **Order** directives as discussed below. To allow only particular hosts or groups of hosts to access the server, the *host* can be specified in any of the following formats:

A (partial) domain-name

Example:

```
Allow from apache.org
Allow from .net example.edu
```

Hosts whose names match, or end in, this string are allowed access. Only complete components are matched, so the above example will match `foo.apache.org` but it will not match `fooapache.org`. This configuration will cause Apache to perform a double reverse

DNS lookup on the client IP address, regardless of the setting of the [HostnameLookups](#) directive. It will do a reverse DNS lookup on the IP address to find the associated hostname, and then do a forward lookup on the hostname to assure that it matches the original IP address. Only if the forward and reverse DNS are consistent and the hostname matches will access be allowed.

A full IP address

Example:

```
Allow from 10.1.2.3  
Allow from 192.168.1.104 192.168.1.205
```

An IP address of a host allowed access

A partial IP address

Example:

```
Allow from 10.1  
Allow from 10 172.20 192.168.2
```

The first 1 to 3 bytes of an IP address, for subnet restriction.

A network/netmask pair

Example:

```
Allow from 10.1.0.0/255.255.0.0
```

A network a.b.c.d, and a netmask w.x.y.z. For more fine-grained subnet restriction.

A network/nnn CIDR specification

Example:

```
Allow from 10.1.0.0/16
```

Similar to the previous case, except the netmask consists of *nnn* high-order 1 bits.

Note that the last three examples above match exactly the same set of hosts.

IPv6 addresses and IPv6 subnets can be specified as shown below:

```
Allow from 2001:db8::a00:20ff:fea7:ccea  
Allow from 2001:db8::a00:20ff:fea7:ccea/10
```

The third format of the arguments to the **Allow** directive allows access to the server to be controlled based on the existence of an [environment variable](#). When `Allow from env=env-variable` is specified, then the request is allowed access if the environment variable *env-variable* exists. When `Allow from env=!env-variable` is specified, then the request is allowed access if the environment variable *env-variable* doesn't exist. The server provides the ability to set environment variables in a flexible way based on characteristics of the client request using the directives provided by [mod_setenvif](#). Therefore, this directive can be used to allow access based on such factors as the clients User - Agent (browser type), Referer, or other HTTP request header fields.

Example:

```
SetEnvIf User-Agent ^KnockKnock/2\.0 let_me_in  
<Directory /docroot>  
    Order Deny,Allow  
    Deny from all  
    Allow from env=let_me_in  
</Directory>
```

In this case, browsers with a user-agent string beginning with `KnockKnock/2.0` will be allowed access, and all others will be denied.



Deny Directive

Description:	Controls which hosts are denied access to the server
Syntax:	Deny from <code>all host env=[!]env-variable</code> <code>[host env=[!]env-variable] ...</code>
Context:	directory, .htaccess
Override:	Limit
Status:	Base
Module:	mod_authz_host

This directive allows access to the server to be restricted based on hostname, IP address, or environment variables. The arguments for the **Deny** directive are identical to the arguments for the **Allow** directive.



Description:	Controls the default access state and the order in which <i>Allow</i> and <i>Deny</i> are evaluated.
Syntax:	Order <i>ordering</i>
Default:	Order Deny,Allow
Context:	directory, .htaccess
Override:	Limit
Status:	Base
Module:	mod_authz_host

The *Order* directive, along with the *Allow* and *Deny* directives, controls a three-pass access control system. The first pass processes either all *Allow* or all *Deny* directives, as specified by the *Order* directive. The second pass parses the rest of the directives (*Deny* or *Allow*). The third pass applies to all requests which do not match either of the first two.

Note that all *Allow* and *Deny* directives are processed, unlike a typical firewall, where only the first match is used. The last match is effective (also unlike a typical firewall). Additionally, the order in which lines appear in the configuration files is not significant -- all *Allow* lines are processed as one group, all *Deny* lines are considered as another, and the default state is considered by itself.

Ordering is one of:

Allow, Deny

First, all *Allow* directives are evaluated; at least one must match, or the request is rejected. Next, all *Deny* directives are evaluated. If any matches, the request is rejected. Last, any requests which do not match an *Allow* or a *Deny* directive are denied by default.

Deny, Allow

First, all *Deny* directives are evaluated; if any match, the request is denied **unless** it also matches an *Allow* directive. Any requests

which do not match any [Allow](#) or [Deny](#) directives are permitted.

Mutual-failure

This order has the same effect as `Order Allow, Deny` and is deprecated in its favor.

Keywords may only be separated by a comma; *no whitespace* is allowed between them.

Match	Allow,Deny result	Deny,Allow result
Match Allow only	Request allowed	Request allowed
Match Deny only	Request denied	Request denied
No match	Default to second directive: Denied	Default to second directive: Allowed
Match both Allow & Deny	Final match controls: Denied	Final match controls: Allowed

In the following example, all hosts in the `apache.org` domain are allowed access; all other hosts are denied access.

```
Order Deny,Allow
Deny from all
Allow from apache.org
```

In the next example, all hosts in the `apache.org` domain are allowed access, except for the hosts which are in the `foo.apache.org` subdomain, who are denied access. All hosts not in the `apache.org` domain are denied access because the default state is to [Deny](#) access to the server.

```
Order Allow,Deny
Allow from apache.org
Deny from foo.apache.org
```

On the other hand, if the [Order](#) in the last example is changed to `Deny, Allow`, all hosts will be allowed access. This happens because,

regardless of the actual ordering of the directives in the configuration file, the `Allow` from `apache.org` will be evaluated last and will override the `Deny` from `foo.apache.org`. All hosts not in the `apache.org` domain will also be allowed access because the default state is [Allow](#).

The presence of an `Order` directive can affect access to a part of the server even in the absence of accompanying [Allow](#) and [Deny](#) directives because of its effect on the default access state. For example,

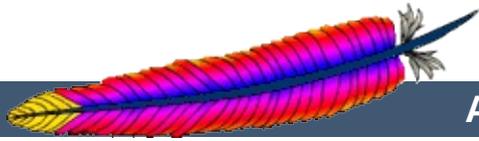
```
<Directory /www>  
  Order Allow,Deny  
</Directory>
```

will `Deny` all access to the `/www` directory because the default access state is set to [Deny](#).

The `Order` directive controls the order of access directive processing only within each phase of the server's configuration processing. This implies, for example, that an [Allow](#) or [Deny](#) directive occurring in a [<Location>](#) section will always be evaluated after an [Allow](#) or [Deny](#) directive occurring in a [<Directory>](#) section or `.htaccess` file, regardless of the setting of the `Order` directive. For details on the merging of configuration sections, see the documentation on [How Directory, Location and Files sections work](#).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authz_owner`

Description:	Authorization based on file ownership
Status:	Extension
Module Identifier:	<code>authz_owner_module</code>
Source File:	<code>mod_authz_owner.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module authorizes access to files by comparing the userid used for HTTP authentication (the web userid) with the file-system owner or group of the requested file. The supplied username and password must be already properly verified by an authentication module, such as [mod_auth_basic](#) or [mod_auth_digest](#). [mod_authz_owner](#) recognizes two arguments for the [Require](#) directive, `file-owner` and `file-group`, as follows:

file-owner

The supplied web-username must match the system's name for the owner of the file being requested. That is, if the operating system says the requested file is owned by jones, then the username used to access it through the web must be jones as well.

file-group

The name of the system group that owns the file must be present in a group database, which is provided, for example, by [mod_authz_groupfile](#) or [mod_authz_dbm](#), and the web-username must be a member of that group. For example, if the operating system says the requested file is owned by (system) group accounts, the group accounts must appear in the group database and the web-username used in the request must be a member of that group.

Note

If [mod_authz_owner](#) is used in order to authorize a resource that is not actually present in the filesystem (*i.e.* a virtual resource), it will deny the access.

Particularly it will never authorize [content negotiated "MultiViews"](#) resources.

See also

[Require](#)

[Satisfy](#)



Require file-owner

Consider a multi-user system running the Apache Web server, with each user having his or her own files in `~/public_html/private`.

Assuming that there is a single [AuthDBMUserFile](#) database that lists all of their web-username, and that these usernames match the system's usernames that actually own the files on the server, then the following stanza would allow only the user himself access to his own files. User `jones` would not be allowed to access files in `/home/smith/public_html/private` unless they were owned by `jones` instead of `smith`.

```
<Directory /home/*/public_html/private>
  AuthType Basic
  AuthName MyPrivateFiles
  AuthBasicProvider dbm
  AuthDBMUserFile /usr/local/apache2/etc/.htdbm-all
  Satisfy All
  Require file-owner
</Directory>
```

Require file-group

Consider a system similar to the one described above, but with some users that share their project files in `~/public_html/project-foo`. The files are owned by the system group `foo` and there is a single [AuthDBMGroupFile](#) database that contains all of the web-username and their group membership, *i.e.* they must be at least member of a group named `foo`. So if `jones` and `smith` are both member of the group `foo`, then both will be authorized to access the `project-foo` directories of each other.

```
<Directory /home/*/public_html/project-foo>
  AuthType Basic
  AuthName "Project Foo Files"
  AuthBasicProvider dbm
```

```
# combined user/group database
AuthDBMUserFile /usr/local/apache2/etc/.htdbm-all
AuthDBMGroupFile /usr/local/apache2/etc/.htdbm-all

Satisfy All
Require file-group
</Directory>
```



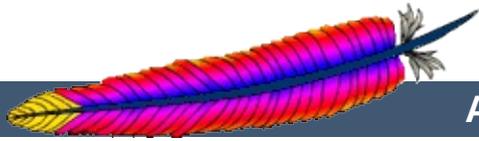
Description:	Sets whether authorization will be passed on to lower level modules
Syntax:	<code>AuthzOwnerAuthoritative On Off</code>
Default:	<code>AuthzOwnerAuthoritative On</code>
Context:	directory, <code>.htaccess</code>
Override:	<code>AuthConfig</code>
Status:	Extension
Module:	<code>mod_authz_owner</code>

Setting the `AuthzOwnerAuthoritative` directive explicitly to `Off` allows for user authorization to be passed on to lower level modules (as defined in the `modules.c` files) if:

- in the case of `file-owner` the file-system owner does not match the supplied web-username or could not be determined, or
- in the case of `file-group` the file-system group does not contain the supplied web-username or could not be determined.

Note that setting the value to `Off` also allows the combination of `file-owner` and `file-group`, so access will be allowed if either one or the other (or both) match.

By default, control is not passed on and an authorization failure will result in an "Authentication Required" reply. Not setting it to `Off` thus keeps the system secure and forces an NCSA compliant behaviour.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_authz_user`

Description:	User Authorization
Status:	Base
Module Identifier:	<code>authz_user_module</code>
Source File:	<code>mod_authz_user.c</code>
Compatibility:	Available in Apache 2.1 and later

Summary

This module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the web site. `mod_authz_user` grants access if the authenticated user is listed in a `Require user` directive. Alternatively `Require valid-user` can be used to grant access to all successfully authenticated users.

See also

[Require](#)

[Satisfy](#)



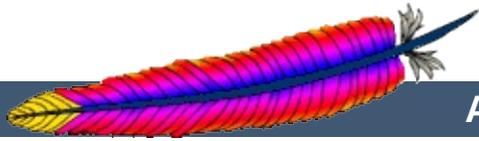
Description:	Sets whether authorization will be passed on to lower level modules
Syntax:	AuthzUserAuthoritative On Off
Default:	AuthzUserAuthoritative On
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	mod_authz_user

Setting the `AuthzUserAuthoritative` directive explicitly to `Off` allows for user authorization to be passed on to lower level modules (as defined in the `modules.c` files) if there is **no user** matching the supplied `userID`.

By default, control is not passed on and an unknown user will result in an Authentication Required reply. Not setting it to `Off` thus keeps the system secure and forces an NCSA compliant behaviour.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_autoindex`

Description:	Generates directory indexes, automatically, similar to the Unix <code>ls</code> command or the Win32 <code>dir</code> shell command
Status:	Base
Module Identifier:	<code>autoindex_module</code>
Source File:	<code>mod_autoindex.c</code>

Summary

The index of a directory can come from one of two sources:

- A file written by the user, typically called `index.html`. The [DirectoryIndex](#) directive sets the name of this file. This is controlled by [mod_dir](#).
- Otherwise, a listing generated by the server. The other directives control the format of this listing. The [AddIcon](#), [AddIconByEncoding](#) and [AddIconByType](#) are used to set a list of icons to display for various file types; for each file listed, the first icon listed that matches the file is displayed. These are controlled by [mod_autoindex](#).

The two functions are separated so that you can completely remove (or replace) automatic index generation should you want to.

Automatic index generation is enabled with using `Options +Indexes`. See the [Options](#) directive for more details.

If the [FancyIndexing](#) option is given with the [IndexOptions](#) directive, the column headers are links that control the order of the display. If you select a header link, the listing will be regenerated, sorted by the values in that column. Selecting the same header repeatedly toggles between ascending and descending order. These

column header links are suppressed with [IndexOptions](#) directive's SuppressColumnSorting option.

Note that when the display is sorted by "Size", it's the *actual* size of the files that's used, not the displayed value - so a 1010-byte file will always be displayed before a 1011-byte file (if in ascending order) even though they both are shown as "1K".



Autoindex Request Query Arguments

Apache 2.0.23 reorganized the Query Arguments for Column Sorting, and introduced an entire group of new query options. To effectively eliminate all client control over the output, the [IndexOptions IgnoreClient](#) option was introduced.

The column sorting headers themselves are self-referencing hyperlinks that add the sort query options shown below. Any option below may be added to any request for the directory resource.

- C=N sorts the directory by file name
- C=M sorts the directory by last-modified date, then file name
- C=S sorts the directory by size, then file name
- C=D sorts the directory by description, then file name

- O=A sorts the listing in Ascending Order
- O=D sorts the listing in Descending Order

- F=0 formats the listing as a simple list (not FancyIndexed)
- F=1 formats the listing as a FancyIndexed list
- F=2 formats the listing as an HTMLTable FancyIndexed list

- V=0 disables version sorting
- V=1 enables version sorting

- P=*pattern* lists only files matching the given *pattern*

Note that the 'P'attern query argument is tested *after* the usual [IndexIgnore](#) directives are processed, and all file names are still subjected to the same criteria as any other autoindex listing. The Query Arguments parser in `mod_autoindex` will stop abruptly when an unrecognized option is encountered. The Query Arguments must be well formed, according to the table above.

The simple example below, which can be clipped and saved in a

header.html file, illustrates these query options. Note that the unknown "X" argument, for the submit button, is listed last to assure the arguments are all parsed before mod_autoindex encounters the X=Go input.

```
<form action="" method="get">
  Show me a <select name="F">
    <option value="0"> Plain list</option>
    <option value="1" selected="selected"> Fancy list</option>
    <option value="2"> Table list</option>
  </select>
  Sorted by <select name="C">
    <option value="N" selected="selected"> Name</option>
    <option value="M"> Date Modified</option>
    <option value="S"> Size</option>
    <option value="D"> Description</option>
  </select>
  <select name="O">
    <option value="A" selected="selected"> Ascending</option>
    <option value="D"> Descending</option>
  </select>
  <select name="V">
    <option value="0" selected="selected"> in Normal order</option>
    <option value="1"> in Version order</option>
  </select>
  Matching <input type="text" name="P" value="" />
  <input type="submit" name="X" value="Go" />
</form>
```



Description: Alternate text to display for a file, instead of an icon selected by filename

Syntax: `AddAlt string file [file] ...`

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

`AddAlt` provides the alternate text to display for a file, instead of an icon, for [FancyIndexing](#). *File* is a file extension, partial filename, wild-card expression or full filename for files to describe. If *String* contains any whitespace, you have to enclose it in quotes (" or '). This alternate text is displayed if the client is image-incapable, has image loading disabled, or fails to retrieve the icon.

Examples

```
AddAlt "PDF file" *.pdf
AddAlt Compressed *.gz *.zip *.Z
```



Description:	Alternate text to display for a file instead of an icon selected by MIME-encoding
Syntax:	<code>AddAltByEncoding <i>string</i> <i>MIME-encoding</i> [<i>MIME-encoding</i>] ...</code>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

`AddAltByEncoding` provides the alternate text to display for a file, instead of an icon, for [FancyIndexing](#). *MIME-encoding* is a valid content-encoding, such as x-compress. If *String* contains any whitespace, you have to enclose it in quotes (" or '). This alternate text is displayed if the client is image-incapable, has image loading disabled, or fails to retrieve the icon.

Example

```
AddAltByEncoding gzip x-gzip
```



Description:	Alternate text to display for a file, instead of an icon selected by MIME content-type
Syntax:	AddAltByType <i>string</i> <i>MIME-type</i> [<i>MIME-type</i>] ...
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

AddAltByType sets the alternate text to display for a file, instead of an icon, for [FancyIndexing](#). *MIME-type* is a valid content-type, such as `text/html`. If *String* contains any whitespace, you have to enclose it in quotes (" or '). This alternate text is displayed if the client is image-incapable, has image loading disabled, or fails to retrieve the icon.

Example

```
AddAltByType 'plain text' text/plain
```



Description:	Description to display for a file
Syntax:	AddDescription <i>string file [file] ...</i>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

This sets the description to display for a file, for [FancyIndexing](#). *File* is a file extension, partial filename, wild-card expression or full filename for files to describe. *String* is enclosed in double quotes (").

Example

```
AddDescription "The planet Mars" mars.gif
AddDescription "My friend Marshall" friends/mars.gif
```

The typical, default description field is 23 bytes wide. 6 more bytes are added by the [IndexOptions SuppressIcon](#) option, 7 bytes are added by the [IndexOptions SuppressSize](#) option, and 19 bytes are added by the [IndexOptions SuppressLastModified](#) option. Therefore, the widest default the description column is ever assigned is 55 bytes.

See the [DescriptionWidth IndexOptions](#) keyword for details on overriding the size of this column, or allowing descriptions of unlimited length.

Caution

Descriptive text defined with [AddDescription](#) may contain HTML markup, such as tags and character entities. If the width of the description column should happen to truncate a tagged element (such as cutting off the end of a bolded phrase), the results may affect the rest of the directory listing.

Arguments with path information

Absolute paths are not currently supported and do not match anything at runtime. Arguments with relative path information, which would normally only be used in htaccess context, are implicitly prefixed with '*' to avoid matching partial directory names.



Description:	Icon to display for a file selected by name
Syntax:	AddIcon <i>icon name</i> [<i>name</i>] ...
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

This sets the icon to display next to a file ending in *name* for [FancyIndexing](#). *Icon* is either a (%-escaped) relative URL to the icon, a fully qualified remote URL, or of the format (*alttext, url*) where *alttext* is the text tag given for an icon for non-graphical browsers.

Name is either `^^DIRECTORY^^` for directories, `^^BLANKICON^^` for blank lines (to format the list correctly), a file extension, a wildcard expression, a partial filename or a complete filename.

`^^BLANKICON^^` is only used for formatting, and so is unnecessary if you're using `IndexOptions HTMLTable`.

Examples

```
AddIcon (IMG,/icons/image.xbm) .gif .jpg .xbm
AddIcon /icons/dir.xbm ^^DIRECTORY^^
AddIcon /icons/backup.xbm *~
```

[AddIconByType](#) should be used in preference to [AddIcon](#), when possible.



Description:	Icon to display next to files selected by MIME content-encoding
Syntax:	<code>AddIconByEncoding <i>icon</i> <i>MIME-encoding</i> [<i>MIME-encoding</i>] ...</code>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

This sets the icon to display next to files with [FancyIndexing](#). *Icon* is either a (%-escaped) relative URL to the icon, a fully qualified remote URL, or of the format (*alt text*, *url*) where *alt text* is the text tag given for an icon for non-graphical browsers.

MIME-encoding is a valid content-encoding, such as x-compress.

Example

```
AddIconByEncoding /icons/compress.xbm x-compress
```



AddIconByType Directive

Description:	Icon to display next to files selected by MIME content-type
Syntax:	AddIconByType <i>icon</i> <i>MIME-type</i> [<i>MIME-type</i>] ...
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

This sets the icon to display next to files of type *MIME-type* for [FancyIndexing](#). *Icon* is either a (%-escaped) relative URL to the icon, a fully qualified remote URL, or of the format (*alttext*, *url*) where *alttext* is the text tag given for an icon for non-graphical browsers.

MIME-type is a wildcard expression matching required the mime types.

Example

```
AddIconByType (IMG,/icons/image.xbm) image/*
```



Description:	Icon to display for files when no specific icon is configured
Syntax:	<code>DefaultIcon <i>url-path</i></code>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

The `DefaultIcon` directive sets the icon to display for files when no specific icon is known, for [FancyIndexing](#). *Url-path* is a (%-escaped) relative URL to the icon, or a fully qualified remote URL.

Example

```
DefaultIcon /icon/unknown.xbm
```



Description:	Name of the file that will be inserted at the top of the index listing
Syntax:	HeaderName <i>filename</i>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

The **HeaderName** directive sets the name of the file that will be inserted at the top of the index listing. *Filename* is the name of the file to include.

Example

```
HeaderName HEADER.html
```

Both **HeaderName** and **ReadmeName** now treat *Filename* as a URI path relative to the one used to access the directory being indexed. If *Filename* begins with a slash, it will be taken to be relative to the **DocumentRoot**.

Example

```
HeaderName /include/HEADER.html
```

Filename must resolve to a document with a major content type of `text/*` (e.g., `text/html`, `text/plain`, etc.). This means that *filename* may refer to a CGI script if the script's actual file type (as opposed to its output) is marked as `text/html` such as with a directive like:

```
AddType text/html .cgi
```

[Content negotiation](#) will be performed if [Options MultiViews](#) is in effect. If *filename* resolves to a static text/html document (not a CGI script) and either one of the [options Includes](#) or [IncludesNOEXEC](#) is enabled, the file will be processed for server-side includes (see the [mod_include](#) documentation).

If the file specified by [HeaderName](#) contains the beginnings of an HTML document (<html>, <head>, etc.) then you will probably want to set [IndexOptions +SuppressHTMLPreamble](#), so that these tags are not repeated.



Description:	Inserts text in the HEAD section of an index page.
Syntax:	<code>IndexHeadInsert "markup ..."</code>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex
Compatibility:	Available in Apache 2.2.11 and later

The `IndexHeadInsert` directive specifies a string to insert in the `<head>` section of the HTML generated for the index page.

Example

```
IndexHeadInsert "<link rel=\"sitemap\" href=\"/sitemap.html\">"
```



Description:	Adds to the list of files to hide when listing a directory
Syntax:	IndexIgnore <i>file</i> [<i>file</i>] ...
Default:	IndexIgnore ". "
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

The **IndexIgnore** directive adds to the list of files to hide when listing a directory. *File* is a shell-style wildcard expression or full filename. Multiple **IndexIgnore** directives add to the list, rather than replacing the list of ignored files. By default, the list contains `.` (the current directory).

```
IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t
```

Regular Expressions

This directive does not currently work in configuration sections that have regular expression arguments, such as [<DirectoryMatch>](#)



Description:	Various configuration settings for directory indexing
Syntax:	<code>IndexOptions [+ -]option [[+ -]option] ...</code>
Default:	By default, no options are enabled.
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	<code>mod_autoindex</code>

The `IndexOptions` directive specifies the behavior of the directory indexing. *Option* can be one of

Charset=*character-set* (Apache 2.0.61 and later)

The `Charset` keyword allows you to specify the character set of the generated page. The default is either *ISO-8859-1* or *UTF-8*, depending on whether the underlying file system is unicode or not.

Example:

```
IndexOptions Charset=UTF-8
```

Type=*MIME content-type* (Apache 2.0.61 and later)

The `Type` keyword allows you to specify the MIME content-type of the generated page. The default is *text/html*.

Example:

```
IndexOptions Type=text/plain
```

DescriptionWidth=[*n* | *] (Apache 2.0.23 and later)

The `DescriptionWidth` keyword allows you to specify the width of the description column in characters.

-`DescriptionWidth` (or `unset`) allows `mod_autoindex` to calculate the best width.

`DescriptionWidth=n` fixes the column width to *n* bytes wide.
`DescriptionWidth=*` grows the column to the width necessary to accommodate the longest description string.

See the section on [AddDescription](#) for dangers inherent in truncating descriptions.

FancyIndexing

This turns on fancy indexing of directories.

FoldersFirst (*Apache 2.0.23 and later*)

If this option is enabled, subdirectory listings will *always* appear first, followed by normal files in the directory. The listing is basically broken into two components, the files and the subdirectories, and each is sorted separately and then displayed subdirectories-first. For instance, if the sort order is descending by name, and `FoldersFirst` is enabled, subdirectory Zed will be listed before subdirectory Beta, which will be listed before normal files Gamma and Alpha. **This option only has an effect if [FancyIndexing](#) is also enabled.**

HTMLTable (*Apache HTTP Server 2.0.23 and later*)

This option with `FancyIndexing` constructs a simple table for the fancy directory listing. It is necessary for utf-8 enabled platforms or if file names or description text will alternate between left-to-right and right-to-left reading order.

IconsAreLinks

This makes the icons part of the anchor for the filename, for fancy indexing.

IconHeight[=*pixels*]

Presence of this option, when used with `IconWidth`, will cause the server to include `height` and `width` attributes in the `img` tag for the file icon. This allows browser to precalculate the page layout without having to wait until all the images have been loaded. If no value is given for the option, it defaults to the standard height of the icons supplied with the Apache software.

IconWidth[=*pixels*]

Presence of this option, when used with `IconHeight`, will cause the server to include `height` and `width` attributes in the `img` tag for the file icon. This allows browser to precalculate the page layout without having to wait until all the images have been loaded. If no value is given for the option, it defaults to the standard width of the icons supplied with the Apache software.

IgnoreCase

If this option is enabled, names are sorted in a case-insensitive manner. For instance, if the sort order is ascending by name, and `IgnoreCase` is enabled, file Zeta will be listed after file alfa (Note: file GAMMA will always be listed before file gamma).

IgnoreClient

This option causes `mod_autoindex` to ignore all query variables from the client, including sort order (implies [SuppressColumnSorting](#).)

NameWidth=[*n* | *]

The `NameWidth` keyword allows you to specify the width of the filename column in bytes.

-`NameWidth` (or `unset`) allows `mod_autoindex` to calculate the best width.

`NameWidth=n` fixes the column width to *n* bytes wide.

`NameWidth=*` grows the column to the necessary width.

ScanHTMLTitles

This enables the extraction of the title from HTML documents for fancy indexing. If the file does not have a description given by [AddDescription](#) then httpd will read the document for the value of the `title` element. This is CPU and disk intensive.

ShowForbidden

If specified, Apache will show files normally hidden because the subrequest returned `HTTP_UNAUTHORIZED` or `HTTP_FORBIDDEN`

SuppressColumnSorting

If specified, Apache will not make the column headings in a FancyIndexed directory listing into links for sorting. The default behavior is for them to be links; selecting the column heading will sort the directory listing by the values in that column. **Prior to Apache 2.0.23, this also disabled parsing the Query Arguments for the sort string.** That behavior is now controlled by [IndexOptions IgnoreClient](#) in Apache 2.0.23.

SuppressDescription

This will suppress the file description in fancy indexing listings. By default, no file descriptions are defined, and so the use of this option will regain 23 characters of screen space to use for something else. See [AddDescription](#) for information about setting the file description. See also the [DescriptionWidth](#) index option to limit the size of the description column.

SuppressHTMLPreamble

If the directory actually contains a file specified by the [HeaderName](#) directive, the module usually includes the contents of the file after a standard HTML preamble (<html>, <head>, *et cetera*). The SuppressHTMLPreamble option disables this behaviour, causing the module to start the display with the header file contents. The header file must contain appropriate HTML instructions in this case. If there is no header file, the preamble is generated as usual.

SuppressIcon (*Apache 2.0.23 and later*)

This will suppress the icon in fancy indexing listings. Combining both SuppressIcon and SuppressRules yields proper HTML 3.2 output, which by the final specification prohibits `img` and `hr` elements from the `pre` block (used to format FancyIndexed listings.)

SuppressLastModified

This will suppress the display of the last modification date, in fancy indexing listings.

SuppressRules (*Apache 2.0.23 and later*)

This will suppress the horizontal rule lines (hr elements) in directory listings. Combining both SuppressIcon and SuppressRules yields proper HTML 3.2 output, which by the final specification prohibits img and hr elements from the pre block (used to format FancyIndexed listings.)

SuppressSize

This will suppress the file size in fancy indexing listings.

TrackModified (*Apache 2.0.23 and later*)

This returns the Last-Modified and ETag values for the listed directory in the HTTP header. It is only valid if the operating system and file system return appropriate stat() results. Some Unix systems do so, as do OS2's JFS and Win32's NTFS volumes. OS2 and Win32 FAT volumes, for example, do not. Once this feature is enabled, the client or proxy can track changes to the list of files when they perform a HEAD request. Note some operating systems correctly track new and removed files, but do not track changes for sizes or dates of the files within the directory. **Changes to the size or date stamp of an existing file will not update the Last-Modified header on all Unix platforms.** If this is a concern, leave this option disabled.

VersionSort (*Apache 2.0a3 and later*)

The VersionSort keyword causes files containing version numbers to sort in a natural way. Strings are sorted as usual, except that substrings of digits in the name and description are compared according to their numeric value.

Example:

```
foo-1.7  
foo-1.7.2  
foo-1.7.12  
foo-1.8.2  
foo-1.8.2a  
foo-1.12
```

If the number starts with a zero, then it is considered to be a fraction:

```
foo-1.001
foo-1.002
foo-1.030
foo-1.04
```

XHTML (*Apache 2.0.49 and later*)

The XHTML keyword forces `mod_autoindex` to emit XHTML 1.0 code instead of HTML 3.2.

Incremental IndexOptions

Apache 1.3.3 introduced some significant changes in the handling of `IndexOptions` directives. In particular:

- Multiple `IndexOptions` directives for a single directory are now merged together. The result of:

```
<Directory /foo>
  IndexOptions HTMLTable
  IndexOptions SuppressColumnsorting
</Directory>
```

will be the equivalent of

```
IndexOptions HTMLTable SuppressColumnsorting
```

- The addition of the incremental syntax (*i.e.*, prefixing keywords with + or -).

Whenever a '+' or '-' prefixed keyword is encountered, it is applied to the current `IndexOptions` settings (which may have been inherited from an upper-level directory). However, whenever an unprefixed keyword is processed, it clears all inherited options and any incremental settings encountered so far. Consider the following example:

```
IndexOptions +ScanHTMLTitles -IconsAreLinks FancyIndexing  
IndexOptions +SuppressSize
```

The net effect is equivalent to `IndexOptions FancyIndexing +SuppressSize`, because the unprefixed `FancyIndexing` discarded the incremental keywords before it, but allowed them to start accumulating again afterward.

To unconditionally set the `IndexOptions` for a particular directory, clearing the inherited settings, specify keywords without any + or - prefixes.



Description:	Sets the default ordering of the directory index
Syntax:	<code>IndexOrderDefault Ascending Descending Name Date Size Description</code>
Default:	<code>IndexOrderDefault Ascending Name</code>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

The `IndexOrderDefault` directive is used in combination with the `FancyIndexing` index option. By default, fancyindexed directory listings are displayed in ascending order by filename; the `IndexOrderDefault` allows you to change this initial display order.

`IndexOrderDefault` takes two arguments. The first must be either `Ascending` or `Descending`, indicating the direction of the sort. The second argument must be one of the keywords `Name`, `Date`, `Size`, or `Description`, and identifies the primary key. The secondary key is *always* the ascending filename.

You can, if desired, prevent the client from reordering the list by also adding the `SuppressColumnSorting` index option to remove the sort link from the top of the column, along with the `IgnoreClient` index option to prevent them from manually adding sort options to the query string in order to override your ordering preferences.



Description:	Adds a CSS stylesheet to the directory index
Syntax:	<code>IndexStyleSheet <i>url-path</i></code>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

The `IndexStyleSheet` directive sets the name of the file that will be used as the CSS for the index listing.

Example

```
IndexStyleSheet "/css/style.css"
```



Description: Name of the file that will be inserted at the end of the index listing

Syntax: ReadmeName *filename*

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

The `ReadmeName` directive sets the name of the file that will be appended to the end of the index listing. *Filename* is the name of the file to include, and is taken to be relative to the location being indexed. If *Filename* begins with a slash, it will be taken to be relative to the `DocumentRoot`.

Example

```
ReadmeName FOOTER.html
```

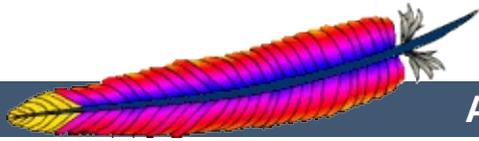
Example 2

```
ReadmeName /include/FOOTER.html
```

See also [HeaderName](#), where this behavior is described in greater detail.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_cache`

Description:	Content cache keyed to URIs.
Status:	Extension
Module Identifier:	<code>cache_module</code>
Source File:	<code>mod_cache.c</code>

Summary

This module should be used with care and can be used to circumvent [Allow](#) and [Deny](#) directives. You should not enable caching for any content to which you wish to limit access by client host name, address or environment variable.

`mod_cache` implements an [RFC 2616](#) compliant HTTP content cache that can be used to cache either local or proxied content. `mod_cache` requires the services of one or more storage management modules. Two storage management modules are included in the base Apache distribution:

`mod_disk_cache`

implements a disk based storage manager.

`mod_mem_cache`

implements a memory based storage manager. `mod_mem_cache` can be configured to operate in two modes: caching open file descriptors or caching objects in heap storage. `mod_mem_cache` can be used to cache locally generated content or to cache backend server content for `mod_proxy` when configured using [ProxyPass](#) (aka *reverse proxy*)

Content is stored in and retrieved from the cache using URI based keys. Content with access protection is not cached.

Further details, discussion, and examples, are provided in the [Caching Guide](#).

See also

[Caching Guide](#)



Related Modules

[mod_disk_cache](#)

[mod_mem_cache](#)

Related Directives

[CacheRoot](#)

[CacheDirLevels](#)

[CacheDirLength](#)

[CacheMinFileSize](#)

[CacheMaxFileSize](#)

[MCacheSize](#)

[MCacheMaxObjectCount](#)

[MCacheMinObjectSize](#)

[MCacheMaxObjectSize](#)

[MCacheRemovalAlgorithm](#)

[MCacheMaxStreamingBuffer](#)



Sample httpd.conf

```
#
# Sample Cache Configuration
#
LoadModule cache_module modules/mod_cache.so

<IfModule mod_cache.c>
  #LoadModule disk_cache_module modules/mod_disk_cache.so
  # If you want to use mod_disk_cache instead of mod_mem_cache,
  # uncomment the line above and comment out the LoadModule line
  # below.
  <IfModule mod_disk_cache.c>
    CacheRoot c:/cacheroot
    CacheEnable disk /
    CacheDirLevels 5
    CacheDirLength 3
  </IfModule>

  LoadModule mem_cache_module modules/mod_mem_cache.so
  <IfModule mod_mem_cache.c>
    CacheEnable mem /
    MCacheSize 4096
    MCacheMaxObjectCount 100
    MCacheMinObjectSize 1
    MCacheMaxObjectSize 2048
  </IfModule>

  # When acting as a proxy, don't cache the list of security updates
  CacheDisable http://security.update.server/update-list/
</IfModule>
```



Preventing the Thundering Herd

When a cached entry becomes stale, `mod_cache` will submit a conditional request to the backend, which is expected to confirm whether the cached entry is still fresh, and send an updated entity if not.

A small but finite amount of time exists between the time the cached entity becomes stale, and the time the stale entity is fully refreshed. On a busy server, a significant number of requests might arrive during this time, and cause a **thundering herd** of requests to strike the backend suddenly and unpredictably.

To keep the thundering herd at bay, the `CacheLock` directive can be used to define a directory in which locks are created for URLs **in flight**. The lock is used as a **hint** by other requests to either suppress an attempt to cache (someone else has gone to fetch the entity), or to indicate that a stale entry is being refreshed (stale content will be returned in the mean time).

Initial caching of an entry

When an entity is cached for the first time, a lock will be created for the entity until the response has been fully cached. During the lifetime of the lock, the cache will suppress the second and subsequent attempt to cache the same entity. While this doesn't hold back the thundering herd, it does stop the cache attempting to cache the same entity multiple times simultaneously.

Refreshment of a stale entry

When an entity reaches its freshness lifetime and becomes stale, a lock will be created for the entity until the response has either been confirmed as still fresh, or replaced by the backend. During the lifetime of the lock, the second and subsequent incoming request will cause stale data to be returned, and the thundering herd is kept at bay.

Locks and Cache-Control: no-cache

Locks are used as a **hint only** to enable the cache to be more gentle on backend servers, however the lock can be overridden if necessary. If the client sends a request with a Cache-Control header forcing a reload, any lock that may be present will be ignored, and the client's request will be honoured immediately and the cached entry refreshed.

As a further safety mechanism, locks have a configurable maximum age. Once this age has been reached, the lock is removed, and a new request is given the opportunity to create a new lock. This maximum age can be set using the `CacheLockMaxAge` directive, and defaults to 5 seconds.

Example configuration

Enabling the cache lock

```
#  
# Enable the cache lock  
#  
<IfModule mod_cache.c>  
    CacheLock on  
    CacheLockPath /tmp/mod_cache-lock  
    CacheLockMaxAge 5  
</IfModule>
```



Description: The default duration to cache a document when no expiry date is specified.

Syntax: `CacheDefaultExpire` *seconds*

Default: `CacheDefaultExpire 3600` (one hour)

Context: server config, virtual host

Status: Extension

Module: `mod_cache`

The `CacheDefaultExpire` directive specifies a default time, in seconds, to cache a document if neither an expiry date nor last-modified date are provided with the document. The value specified with the `CacheMaxExpire` directive does *not* override this setting.

```
CacheDefaultExpire 86400
```



Description:	Disable caching of specified URLs
Syntax:	CacheDisable <i>url-string</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache

The `CacheDisable` directive instructs `mod_cache` to *not* cache urls at or below *url-string*.

Example

```
CacheDisable /local_files
```

The `no-cache` environment variable can be set to disable caching on a finer grained set of resources in versions 2.2.12 and later.

See also

- [Environment Variables in Apache](#)



Description: Enable caching of specified URLs using a specified storage manager

Syntax: `CacheEnable cache_type url-string`

Context: server config, virtual host

Status: Extension

Module: `mod_cache`

The `CacheEnable` directive instructs `mod_cache` to cache urls at or below *url-string*. The cache storage manager is specified with the *cache_type* argument. *cache_type* `mem` instructs `mod_cache` to use the memory based storage manager implemented by `mod_mem_cache`. *cache_type* `disk` instructs `mod_cache` to use the disk based storage manager implemented by `mod_disk_cache`. *cache_type* `fd` instructs `mod_cache` to use the file descriptor cache implemented by `mod_mem_cache`.

In the event that the URL space overlaps between different `CacheEnable` directives (as in the example below), each possible storage manager will be run until the first one that actually processes the request. The order in which the storage managers are run is determined by the order of the `CacheEnable` directives in the configuration file.

```
CacheEnable mem /manual
CacheEnable fd /images
CacheEnable disk /
```

When acting as a forward proxy server, *url-string* can also be used to specify remote sites and proxy protocols which caching should be enabled for.

```
# Cache proxied url's
CacheEnable disk /

# Cache FTP-proxied url's
```

```
CacheEnable disk ftp://  
  
# Cache content from www.apache.org  
CacheEnable disk http://www.apache.org/
```

The no-cache environment variable can be set to disable caching on a finer grained set of resources in versions 2.2.12 and later.

See also

- [Environment Variables in Apache](#)



Description:	Ignore request to not serve cached content to client
Syntax:	CacheIgnoreCacheControl On Off
Default:	CacheIgnoreCacheControl Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache

Ordinarily, requests containing a Cache-Control: no-cache or Pragma: no-cache header value will not be served from the cache. The `CacheIgnoreCacheControl` directive allows this behavior to be overridden. `CacheIgnoreCacheControl On` tells the server to attempt to serve the resource from the cache even if the request contains no-cache header values. Resources requiring authorization will *never* be cached.

```
CacheIgnoreCacheControl On
```

Warning:

This directive will allow serving from the cache even if the client has requested that the document not be served from the cache. This might result in stale content being served.

See also

- [CacheStorePrivate](#)
- [CacheStoreNoStore](#)



Description:	Do not store the given HTTP header(s) in the cache.
Syntax:	CacheIgnoreHeaders <i>header-string</i> [<i>header-string</i>] ...
Default:	CacheIgnoreHeaders None
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache

According to RFC 2616, hop-by-hop HTTP headers are not stored in the cache. The following HTTP headers are hop-by-hop headers and thus do not get stored in the cache in *any* case regardless of the setting of **CacheIgnoreHeaders**:

- Connection
- Keep-Alive
- Proxy-Authenticate
- Proxy-Authorization
- TE
- Trailers
- Transfer-Encoding
- Upgrade

CacheIgnoreHeaders specifies additional HTTP headers that should not to be stored in the cache. For example, it makes sense in some cases to prevent cookies from being stored in the cache.

CacheIgnoreHeaders takes a space separated list of HTTP headers that should not be stored in the cache. If only hop-by-hop headers not should be stored in the cache (the RFC 2616 compliant behaviour), **CacheIgnoreHeaders** can be set to None.

Example 1

CacheIgnoreHeaders Set-Cookie

Example 2

CacheIgnoreHeaders None

Warning:

If headers like Expires which are needed for proper cache management are not stored due to a CacheIgnoreHeaders setting, the behaviour of mod_cache is undefined.



Description: Ignore the fact that a response has no Last Modified header.

Syntax: CacheIgnoreNoLastMod On|Off

Default: CacheIgnoreNoLastMod Off

Context: server config, virtual host

Status: Extension

Module: mod_cache

Ordinarily, documents without a last-modified date are not cached. Under some circumstances the last-modified date is removed (during `mod_include` processing for example) or not provided at all. The `CacheIgnoreNoLastMod` directive provides a way to specify that documents without last-modified dates should be considered for caching, even without a last-modified date. If neither a last-modified date nor an expiry date are provided with the document then the value specified by the `CacheDefaultExpire` directive will be used to generate an expiration date.

```
CacheIgnoreNoLastMod On
```



CacheIgnoreQueryString Directive

Description:	Ignore query string when caching
Syntax:	CacheIgnoreQueryString On Off
Default:	CacheIgnoreQueryString Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache
Compatibility:	Available in Apache 2.2.6 and later

Ordinarily, requests with query string parameters are cached separately for each unique query string. This is according to RFC 2616/13.9 done only if an expiration time is specified. The **CacheIgnoreQueryString** directive tells the cache to cache requests even if no expiration time is specified, and to reply with a cached reply even if the query string differs. From a caching point of view the request is treated as if having no query string when this directive is enabled.

```
CacheIgnoreQueryString On
```



Description:	Ignore defined session identifiers encoded in the URL when caching
Syntax:	<code>CacheIgnoreURLSessionIdentifiers</code> <i>identifier</i> [<i>identifier</i>] ...
Default:	<code>CacheIgnoreURLSessionIdentifiers</code> None
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache

Sometimes applications encode the session identifier into the URL like in the following Examples:

- `/someapplication/image.gif;jsessionid=123456789`
- `/someapplication/image.gif?PHPSESSIONID=12345678`

This causes cachable resources to be stored separately for each session, which is often not desired.

`CacheIgnoreURLSessionIdentifiers` lets define a list of identifiers that are removed from the key that is used to identify an entity in the cache, such that cachable resources are not stored separately for each session.

`CacheIgnoreURLSessionIdentifiers` None clears the list of ignored identifiers. Otherwise, each identifier is added to the list.

Example 1

```
CacheIgnoreURLSessionIdentifiers jsessionid
```

Example 2

```
CacheIgnoreURLSessionIdentifiers None
```



Description: The factor used to compute an expiry date based on the LastModified date.

Syntax: CacheLastModifiedFactor *float*

Default: CacheLastModifiedFactor 0.1

Context: server config, virtual host

Status: Extension

Module: mod_cache

In the event that a document does not provide an expiry date but does provide a last-modified date, an expiry date can be calculated based on the time since the document was last modified. The `CacheLastModifiedFactor` directive specifies a *factor* to be used in the generation of this expiry date according to the following formula:
$$\text{expiry-period} = \text{time-since-last-modified-date} * \text{factor}$$
$$\text{expiry-date} = \text{current-date} + \text{expiry-period}$$
For example, if the document was last modified 10 hours ago, and *factor* is 0.1 then the expiry-period will be set to $10 * 0.1 = 1$ hour. If the current time was 3:00pm then the computed expiry-date would be 3:00pm + 1hour = 4:00pm. If the expiry-period would be longer than that set by `CacheMaxExpire`, then the latter takes precedence.

```
CacheLastModifiedFactor 0.5
```



Description:	Enable the thundering herd lock.
Syntax:	CacheLock <i>on off</i>
Default:	CacheLock off
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache
Compatibility:	Available in Apache 2.2.15 and later

The **CacheLock** directive enables the thundering herd lock for the given URL space.

In a minimal configuration the following directive is all that is needed to enable the thundering herd lock in the default system temp directory.

```
# Enable chache lock  
CacheLock on
```



Description:	Set the maximum possible age of a cache lock.
Syntax:	CacheLockMaxAge <i>integer</i>
Default:	CacheLockMaxAge 5
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache

The **CacheLockMaxAge** directive specifies the maximum age of any cache lock.

A lock older than this value in seconds will be ignored, and the next incoming request will be given the opportunity to re-establish the lock. This mechanism prevents a slow client taking an excessively long time to refresh an entity.



Description:	Set the lock path directory.
Syntax:	CacheLockPath <i>directory</i>
Default:	CacheLockPath /tmp/mod_cache-lock
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache

The **CacheLockPath** directive allows you to specify the directory in which the locks are created. By default, the system's temporary folder is used. Locks consist of empty files that only exist for stale URLs in flight, so is significantly less resource intensive than the traditional disk cache.



CacheMaxExpire Directive

Description:	The maximum time in seconds to cache a document
Syntax:	CacheMaxExpire <i>seconds</i>
Default:	CacheMaxExpire 86400 (one day)
Context:	server config, virtual host
Status:	Extension
Module:	mod_cache

The `CacheMaxExpire` directive specifies the maximum number of seconds for which cachable HTTP documents will be retained without checking the origin server. Thus, documents will be out of date at most this number of seconds. This maximum value is enforced even if an expiry date was supplied with the document.

```
CacheMaxExpire 604800
```



Description: Attempt to cache requests or responses that have been marked as no-store.

Syntax: CacheStoreNoStore On|Off

Default: CacheStoreNoStore Off

Context: server config, virtual host

Status: Extension

Module: mod_cache

Ordinarily, requests or responses with Cache-Control: no-store header values will not be stored in the cache. The `CacheStoreNoStore` directive allows this behavior to be overridden. `CacheStoreNoStore On` tells the server to attempt to cache the resource even if it contains no-store header values. Resources requiring authorization will *never* be cached.

```
CacheStoreNoStore On
```

Warning:

As described in RFC 2616, the no-store directive is intended to "prevent the inadvertent release or retention of sensitive information (for example, on backup tapes)." Enabling this option could store sensitive information in the cache. You are hereby warned.

See also

- [CacheIgnoreCacheControl](#)
- [CacheStorePrivate](#)



Description: Attempt to cache responses that the server has marked as private

Syntax: `CacheStorePrivate On|Off`

Default: `CacheStorePrivate Off`

Context: server config, virtual host

Status: Extension

Module: `mod_cache`

Ordinarily, responses with `Cache-Control: private` header values will not be stored in the cache. The `CacheStorePrivate` directive allows this behavior to be overridden. `CacheStorePrivate On` tells the server to attempt to cache the resource even if it contains private header values. Resources requiring authorization will *never* be cached.

```
CacheStorePrivate On
```

Warning:

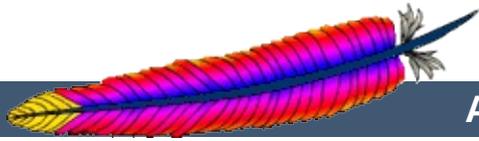
This directive will allow caching even if the upstream server has requested that the resource not be cached. This directive is only ideal for a 'private' cache.

See also

- [CacheIgnoreCacheControl](#)
- [CacheStoreNoStore](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_cern_meta

Description:	CERN httpd metafile semantics
Status:	Extension
Module Identifier:	cern_meta_module
Source File:	mod_cern_meta.c

Summary

Emulate the CERN HTTPD Meta file semantics. Meta files are HTTP headers that can be output in addition to the normal range of headers for each file accessed. They appear rather like the Apache .asis files, and are able to provide a crude way of influencing the Expires: header, as well as providing other curiosities. There are many ways to manage meta information, this one was chosen because there is already a large number of CERN users who can exploit this module.

More information on the [CERN metafile semantics](#) is available.

See also

[mod_headers](#)

[mod_asis](#)



Description:	Name of the directory to find CERN-style meta information files
Syntax:	MetaDir <i>directory</i>
Default:	MetaDir .web
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Extension
Module:	mod_cern_meta

Specifies the name of the directory in which Apache can find meta information files. The directory is usually a 'hidden' subdirectory of the directory that contains the file being accessed. Set to "." to look in the same directory as the file:

```
MetaDir .
```

Or, to set it to a subdirectory of the directory containing the files:

```
MetaDir .meta
```



Description:	Activates CERN meta-file processing
Syntax:	MetaFiles on off
Default:	MetaFiles off
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Extension
Module:	mod_cern_meta

Turns on/off Meta file processing on a per-directory basis.



Description:	File name suffix for the file containing CERN-style meta information
Syntax:	MetaSuffix <i>suffix</i>
Default:	MetaSuffix .meta
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Extension
Module:	mod_cern_meta

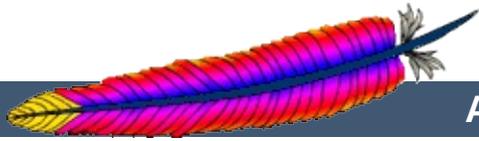
Specifies the file name suffix for the file containing the meta information. For example, the default values for the two directives will cause a request to DOCUMENT_ROOT/somedir/index.html to look in DOCUMENT_ROOT/somedir/.web/index.html.meta and will use its contents to generate additional MIME header information.

Example:

```
MetaSuffix .meta
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_cgi`

Description:	Execution of CGI scripts
Status:	Base
Module Identifier:	<code>cgi_module</code>
Source File:	<code>mod_cgi.c</code>

Summary

Any file that has the handler `cgi-script` will be treated as a CGI script, and run by the server, with its output being returned to the client. Files acquire this handler either by having a name containing an extension defined by the [AddHandler](#) directive, or by being in a [ScriptAlias](#) directory.

For an introduction to using CGI scripts with Apache, see our tutorial on [Dynamic Content With CGI](#).

When using a multi-threaded MPM under unix, the module [mod_cgid](#) should be used in place of this module. At the user level, the two modules are essentially identical.

For backward-compatibility, the `cgi-script` handler will also be activated for any file with the mime-type `application/x-httpd-cgi`. The use of the magic mime-type is deprecated.

See also

[AcceptPathInfo](#)

[Options](#)

[ScriptAlias](#)

[AddHandler](#)

[Running CGI programs under different user IDs](#)

CGI Specification



The server will set the CGI environment variables as described in the [CGI specification](#), with the following provisions:

PATH_INFO

This will not be available if the [AcceptPathInfo](#) directive is explicitly set to off. The default behavior, if [AcceptPathInfo](#) is not given, is that [mod_cgi](#) will accept path info (trailing /more/path/info following the script filename in the URI), while the core server will return a 404 NOT FOUND error for requests with additional path info. Omitting the [AcceptPathInfo](#) directive has the same effect as setting it On for [mod_cgi](#) requests.

REMOTE_HOST

This will only be set if [HostnameLookups](#) is set to on (it is off by default), and if a reverse DNS lookup of the accessing host's address indeed finds a host name.

REMOTE_IDENT

This will only be set if [IdentityCheck](#) is set to on and the accessing host supports the ident protocol. Note that the contents of this variable cannot be relied upon because it can easily be faked, and if there is a proxy between the client and the server, it is usually totally useless.

REMOTE_USER

This will only be set if the CGI script is subject to authentication.

This module also leverages the core functions [ap_add_common_vars](#) and [ap_add_cgi_vars](#) to add environment variables like:

DOCUMENT_ROOT

Set with the content of the related [DocumentRoot](#) directive.

SERVER_NAME

The fully qualified domain name related to the request.

SERVER_ADDR

The IP address of the Virtual Host serving the request.

SERVER_ADMIN

Set with the content of the related [ServerAdmin](#) directive.

For an exhaustive list it is suggested to write a basic CGI script that dumps all the environment variables passed by Apache in a convenient format.



CGI Debugging

Debugging CGI scripts has traditionally been difficult, mainly because it has not been possible to study the output (standard output and error) for scripts which are failing to run properly. These directives, included in Apache 1.2 and later, provide more detailed logging of errors when they occur.

CGI Logfile Format

When configured, the CGI error log logs any CGI which does not execute properly. Each CGI script which fails to operate causes several lines of information to be logged. The first two lines are always of the format:

```
%% [time] request-line
%% HTTP-status CGI-script-filename
```

If the error is that CGI script cannot be run, the log file will contain an extra two lines:

```
%%error
error-message
```

Alternatively, if the error is the result of the script returning incorrect header information (often due to a bug in the script), the following information is logged:

```
%request
All HTTP request headers received
POST or PUT entity (if any)
%response
All headers output by the CGI script
%stdout
CGI standard output
%stderr
CGI standard error
```

(The %stdout and %stderr parts may be missing if the script did not output anything on standard output or standard error).



Description:	Location of the CGI script error logfile
Syntax:	ScriptLog <i>file-path</i>
Context:	server config, virtual host
Status:	Base
Module:	mod_cgi , mod_cgid

The **ScriptLog** directive sets the CGI script error logfile. If no **ScriptLog** is given, no error log is created. If given, any CGI errors are logged into the filename given as argument. If this is a relative file or path it is taken relative to the **ServerRoot**.

Example

```
ScriptLog logs/cgi_log
```

This log will be opened as the user the child processes run as, *i.e.* the user specified in the main **User** directive. This means that either the directory the script log is in needs to be writable by that user or the file needs to be manually created and set to be writable by that user. If you place the script log in your main logs directory, do **NOT** change the directory permissions to make it writable by the user the child processes run as.

Note that script logging is meant to be a debugging feature when writing CGI scripts, and is not meant to be activated continuously on running servers. It is not optimized for speed or efficiency, and may have security problems if used in a manner other than that for which it was designed.



Description:	Maximum amount of PUT or POST requests that will be recorded in the scriptlog
Syntax:	ScriptLogBuffer <i>bytes</i>
Default:	ScriptLogBuffer 1024
Context:	server config, virtual host
Status:	Base
Module:	mod_cgi , mod_cgid

The size of any PUT or POST entity body that is logged to the file is limited, to prevent the log file growing too big too quickly if large bodies are being received. By default, up to 1024 bytes are logged, but this can be changed with this directive.

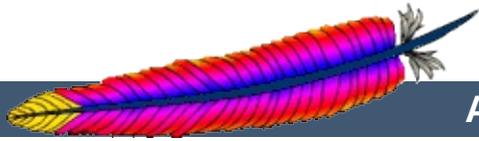


Description:	Size limit of the CGI script logfile
Syntax:	ScriptLogLength <i>bytes</i>
Default:	ScriptLogLength 10385760
Context:	server config, virtual host
Status:	Base
Module:	mod_cgi , mod_cgid

ScriptLogLength can be used to limit the size of the CGI script logfile. Since the logfile logs a lot of information per CGI error (all request headers, all script output) it can grow to be a big file. To prevent problems due to unbounded growth, this directive can be used to set an maximum file-size for the CGI logfile. If the file exceeds this size, no more information will be written to it.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_cgid`

Description:	Execution of CGI scripts using an external CGI daemon
Status:	Base
Module Identifier:	<code>cgid_module</code>
Source File:	<code>mod_cgid.c</code>
Compatibility:	Unix threaded MPMs only

Summary

Except for the optimizations and the additional [ScriptSock](#) directive noted below, `mod_cgid` behaves similarly to `mod_cgi`. See the [mod_cgi](#) summary for additional details about Apache and CGI.

On certain unix operating systems, forking a process from a multi-threaded server is a very expensive operation because the new process will replicate all the threads of the parent process. In order to avoid incurring this expense on each CGI invocation, `mod_cgid` creates an external daemon that is responsible for forking child processes to run CGI scripts. The main server communicates with this daemon using a unix domain socket.

This module is used by default instead of `mod_cgi` whenever a multi-threaded MPM is selected during the compilation process. At the user level, this module is identical in configuration and operation to `mod_cgi`. The only exception is the additional directive `ScriptSock` which gives the name of the socket to use for communication with the cgi daemon.

See also

[mod_cgi](#)

Running CGI programs under different user IDs



Description:	The length of time to wait for more output from the CGI program
Syntax:	CGIDScriptTimeout <i>time</i> [s ms]
Default:	value of Timeout directive when unset
Context:	server config, virtual host, directory, .htaccess
Status:	Base
Module:	mod_cgid
Compatibility:	CGIDScriptTimeout defaults to zero in releases 2.4 and earlier

This directive limits the length of time to wait for more output from the CGI program. If the time is exceeded, the request and CGI are terminated.

Example

```
CGIDScriptTimeout 20
```



Description: The filename prefix of the socket to use for communication with the cgi daemon

Syntax: ScriptSock *file-path*

Default: ScriptSock logs/cgisock

Context: server config

Status: Base

Module: mod_cgid

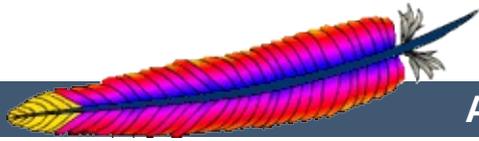
This directive sets the filename prefix of the socket to use for communication with the CGI daemon, an extension corresponding to the process ID of the server will be appended. The socket will be opened using the permissions of the user who starts Apache (usually root). To maintain the security of communications with CGI scripts, it is important that no other user has permission to write in the directory where the socket is located.

Example

```
ScriptSock /var/run/cgid.sock
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_charset_lite`

Description:	Specify character set translation or recoding
Status:	Extension
Module Identifier:	<code>charset_lite_module</code>
Source File:	<code>mod_charset_lite.c</code>

Summary

`mod_charset_lite` allows the server to change the character set of responses before sending them to the client. In an EBCDIC environment, Apache always translates HTTP protocol content (e.g. response headers) from the code page of the Apache process locale to ISO-8859-1, but not the body of responses. In any environment, `mod_charset_lite` can be used to specify that response bodies should be translated. For example, if files are stored in EBCDIC, then `mod_charset_lite` can translate them to ISO-8859-1 before sending them to the client.

This module provides a small subset of configuration mechanisms implemented by Russian Apache and its associated `mod_charset`.



Invalid character set names

The character set name parameters of `CharsetSourceEnc` and `CharsetDefault` must be acceptable to the translation mechanism used by `APR` on the system where `mod_charset_lite` is deployed. These character set names are not standardized and are usually not the same as the corresponding values used in http headers. Currently, `APR` can only use `iconv(3)`, so you can easily test your character set names using the `iconv(1)` program, as follows:

```
iconv -f charsetsourceenc-value -t charsetdefault-value
```

Mismatch between character set of content and translation rules

If the translation rules don't make sense for the content, translation can fail in various ways, including:

- The translation mechanism may return a bad return code, and the connection will be aborted.
- The translation mechanism may silently place special characters (e.g., question marks) in the output buffer when it cannot translate the input buffer.



Description:	Charset to translate into
Syntax:	CharsetDefault <i>charset</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_charset_lite

The `CharsetDefault` directive specifies the charset that content in the associated container should be translated to.

The value of the *charset* argument must be accepted as a valid character set name by the character set support in [APR](#). Generally, this means that it must be supported by iconv.

Example

```
<Directory /export/home/trawick/apacheinst/htdocs/convert>
  CharsetSourceEnc UTF-16BE
  CharsetDefault ISO-8859-1
</Directory>
```



Description:	Configures charset translation behavior
Syntax:	CharsetOptions <i>option</i> [<i>option</i>] ...
Default:	CharsetOptions DebugLevel=0 NoImplicitAdd
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_charset_lite

The `CharsetOptions` directive configures certain behaviors of `mod_charset_lite`. *Option* can be one of

DebugLevel=*n*

The `DebugLevel` keyword allows you to specify the level of debug messages generated by `mod_charset_lite`. By default, no messages are generated. This is equivalent to `DebugLevel=0`. With higher numbers, more debug messages are generated, and server performance will be degraded. The actual meanings of the numeric values are described with the definitions of the `DBGLVL_` constants near the beginning of `mod_charset_lite.c`.

ImplicitAdd | NoImplicitAdd

The `ImplicitAdd` keyword specifies that `mod_charset_lite` should implicitly insert its filter when the configuration specifies that the character set of content should be translated. If the filter chain is explicitly configured using the `AddOutputFilter` directive, `NoImplicitAdd` should be specified so that `mod_charset_lite` doesn't add its filter.

TranslateAllMimeTypes | NoTranslateAllMimeTypes

Normally, `mod_charset_lite` will only perform translation on a small subset of possible mimetypes. When the `TranslateAllMimeTypes` keyword is specified for a given configuration section, translation is performed without regard for

mimetype.



Description:	Source charset of files
Syntax:	CharsetSourceEnc <i>charset</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_charset_lite

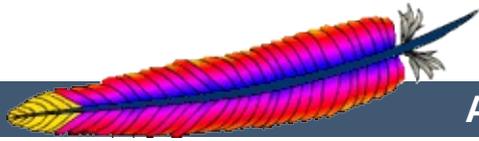
The `CharsetSourceEnc` directive specifies the source charset of files in the associated container.

The value of the *charset* argument must be accepted as a valid character set name by the character set support in [APR](#). Generally, this means that it must be supported by iconv.

Example

```
<Directory /export/home/trawick/apacheinst/htdocs/convert>  
  CharsetSourceEnc UTF-16BE  
  CharsetDefault ISO-8859-1  
</Directory>
```

The character set names in this example work with the iconv translation support in Solaris 8.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_dav

Description:	Distributed Authoring and Versioning (WebDAV) functionality
Status:	Extension
Module Identifier:	dav_module
Source File:	mod_dav.c

Summary

This module provides class 1 and class 2 [WebDAV](#) ('Web-based Distributed Authoring and Versioning') functionality for Apache. This extension to the HTTP protocol allows creating, moving, copying, and deleting resources and collections on a remote web server.

See also

[DavLockDB](#)

[LimitXMLRequestBody](#)

[WebDAV Resources](#)



To enable `mod_dav`, add the following to a container in your `httpd.conf` file:

```
Dav On
```

This enables the DAV file system provider, which is implemented by the `mod_dav_fs` module. Therefore, that module must be compiled into the server or loaded at runtime using the `LoadModule` directive.

In addition, a location for the DAV lock database must be specified in the global section of your `httpd.conf` file using the `DavLockDB` directive:

```
DavLockDB /usr/local/apache2/var/DavLock
```

The directory containing the lock database file must be writable by the `User` and `Group` under which Apache is running.

You may wish to add a `<Limit>` clause inside the `<Location>` directive to limit access to DAV-enabled locations. If you want to set the maximum amount of bytes that a DAV client can send at one request, you have to use the `LimitXMLRequestBody` directive. The "normal" `LimitRequestBody` directive has no effect on DAV requests.

Full Example

```
DavLockDB /usr/local/apache2/var/DavLock
```

```
<Location /foo>
  Order Allow,Deny
  Allow from all
  Dav On

  AuthType Basic
  AuthName DAV
  AuthUserFile user.passwd

  <LimitExcept GET OPTIONS>
    Require user admin
```

```
</LimitExcept>  
</Location>
```

[mod_dav](#) is a descendent of Greg Stein's [mod_dav for Apache 1.3](#). More information about the module is available from that site.



Since DAV access methods allow remote clients to manipulate files on the server, you must take particular care to assure that your server is secure before enabling [mod_dav](#).

Any location on the server where DAV is enabled should be protected by authentication. The use of HTTP Basic Authentication is not recommended. You should use at least HTTP Digest Authentication, which is provided by the [mod_auth_digest](#) module. Nearly all WebDAV clients support this authentication method. An alternative is Basic Authentication over an [SSL](#) enabled connection.

In order for [mod_dav](#) to manage files, it must be able to write to the directories and files under its control using the [User](#) and [Group](#) under which Apache is running. New files created will also be owned by this [User](#) and [Group](#). For this reason, it is important to control access to this account. The DAV repository is considered private to Apache; modifying files outside of Apache (for example using FTP or filesystem-level tools) should not be allowed.

[mod_dav](#) may be subject to various kinds of denial-of-service attacks. The [LimitXMLRequestBody](#) directive can be used to limit the amount of memory consumed in parsing large DAV requests. The [DavDepthInfinity](#) directive can be used to prevent PROPFIND requests on a very large repository from consuming large amounts of memory. Another possible denial-of-service attack involves a client simply filling up all available disk space with many large files. There is no direct way to prevent this in Apache, so you should avoid giving DAV access to untrusted users.



Complex Configurations

One common request is to use `mod_dav` to manipulate dynamic files (PHP scripts, CGI scripts, etc). This is difficult because a GET request will always run the script, rather than downloading its contents. One way to avoid this is to map two different URLs to the content, one of which will run the script, and one of which will allow it to be downloaded and manipulated with DAV.

```
Alias /phparea /home/gstein/php_files
Alias /php-source /home/gstein/php_files
<Location /php-source>
    DAV On
    ForceType text/plain
</Location>
```

With this setup, `http://example.com/phparea` can be used to access the output of the PHP scripts, and `http://example.com/php-source` can be used with a DAV client to manipulate them.



Description:	Enable WebDAV HTTP methods
Syntax:	Dav On Off <i>provider-name</i>
Default:	Dav Off
Context:	directory
Status:	Extension
Module:	mod_dav

Use the **Dav** directive to enable the WebDAV HTTP methods for the given container:

```
<Location /foo>
  Dav On
</Location>
```

The value **On** is actually an alias for the default provider `filesystem` which is served by the `mod_dav_fs` module. Note, that once you have DAV enabled for some location, it *cannot* be disabled for sublocations. For a complete configuration example have a look at the [section above](#).

Do not enable WebDAV until you have secured your server. Otherwise everyone will be able to distribute files on your system.



Description:	Allow PROPFIND, Depth: Infinity requests
Syntax:	DavDepthInfinity on off
Default:	DavDepthInfinity off
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_dav

Use the `DavDepthInfinity` directive to allow the processing of PROPFIND requests containing the header 'Depth: Infinity'. Because this type of request could constitute a denial-of-service attack, by default it is not allowed.



Description:	Minimum amount of time the server holds a lock on a DAV resource
Syntax:	<code>DavMinTimeout</code> <i>seconds</i>
Default:	<code>DavMinTimeout</code> 0
Context:	server config, virtual host, directory
Status:	Extension
Module:	<code>mod_dav</code>

When a client requests a DAV resource lock, it can also specify a time when the lock will be automatically removed by the server. This value is only a request, and the server can ignore it or inform the client of an arbitrary value.

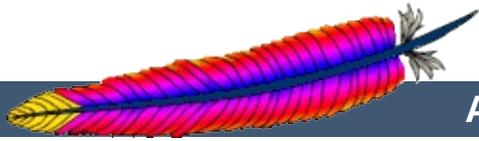
Use the `DavMinTimeout` directive to specify, in seconds, the minimum lock timeout to return to a client. Microsoft Web Folders defaults to a timeout of 120 seconds; the `DavMinTimeout` can override this to a higher value (like 600 seconds) to reduce the chance of the client losing the lock due to network latency.

Example

```
<Location /MSWord>
  DavMinTimeout 600
</Location>
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_dav_fs`

Description:	filesystem provider for <code>mod_dav</code>
Status:	Extension
Module Identifier:	<code>dav_fs_module</code>
Source File:	<code>mod_dav_fs.c</code>

Summary

This module *requires* the service of `mod_dav`. It acts as a support module for `mod_dav` and provides access to resources located in the server's file system. The formal name of this provider is `filesystem`. `mod_dav` backend providers will be invoked by using the `Dav` directive:

Example

```
Dav filesystem
```

Since `filesystem` is the default provider for `mod_dav`, you may simply use the value `On` instead.

See also

[mod_dav](#)



Description:	Location of the DAV lock database
Syntax:	DavLockDB <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_dav_fs

Use the **DavLockDB** directive to specify the full path to the lock database, excluding an extension. If the path is not absolute, it will be taken relative to **ServerRoot**. The implementation of **mod_dav_fs** uses a SDBM database to track user locks.

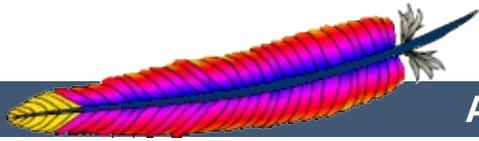
Example

```
DavLockDB var/DavLock
```

The directory containing the lock database file must be writable by the **User** and **Group** under which Apache is running. For security reasons, you should create a directory for this purpose rather than changing the permissions on an existing directory. In the above example, Apache will create files in the `var/` directory under the **ServerRoot** with the base filename `DavLock` and extension name chosen by the server.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_dav_lock`

Description:	generic locking module for <code>mod_dav</code>
Status:	Extension
Module Identifier:	<code>dav_lock_module</code>
Source File:	<code>mod_dav_lock.c</code>
Compatibility:	Available in version 2.1 and later

Summary

This module implements a generic locking API which can be used by any backend provider of `mod_dav`. It *requires* at least the service of `mod_dav`. But without a backend provider which makes use of it, it's useless and should not be loaded into the server. A sample backend module which actually utilizes `mod_dav_lock` is `mod_dav_svn`, the subversion provider module.

Note that `mod_dav_fs` does *not* need this generic locking module, because it uses its own more specialized version.

In order to make `mod_dav_lock` functional, you just have to specify the location of the lock database using the `DavGenericLockDB` directive described below.

Developer's Note

In order to retrieve the pointer to the locking provider function, you have to use the `ap_lookup_provider` API with the arguments `dav-lock`, `generic`, and `0`.

See also

[mod_dav](#)



Description:	Location of the DAV lock database
Syntax:	<code>DavGenericLockDB</code> <i>file-path</i>
Context:	server config, virtual host, directory
Status:	Extension
Module:	<code>mod_dav_lock</code>

Use the `DavGenericLockDB` directive to specify the full path to the lock database, excluding an extension. If the path is not absolute, it will be interpreted relative to `ServerRoot`. The implementation of `mod_dav_lock` uses a SDBM database to track user locks.

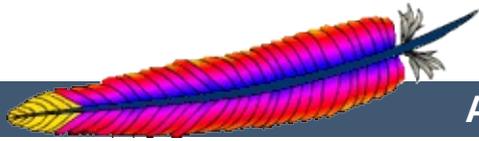
Example

```
DavGenericLockDB var/DavLock
```

The directory containing the lock database file must be writable by the `User` and `Group` under which Apache is running. For security reasons, you should create a directory for this purpose rather than changing the permissions on an existing directory. In the above example, Apache will create files in the `var /` directory under the `ServerRoot` with the base filename `DavLock` and an extension added by the server.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_dbd

Description:	Manages SQL database connections
Status:	Extension
Module Identifier:	dbd_module
Source File:	mod_dbd.c
Compatibility:	Version 2.1 and later

Summary

[mod_dbd](#) manages SQL database connections using [APR](#). It provides database connections on request to modules requiring SQL database functions, and takes care of managing databases with optimal efficiency and scalability for both threaded and non-threaded MPMs. For details, see the [APR](#) website and this overview of the [Apache DBD Framework](#) by its original developer.

See also

[Password Formats](#)



This module manages database connections, in a manner optimised for the platform. On non-threaded platforms, it provides a persistent connection in the manner of classic LAMP (Linux, Apache, Mysql, Perl/PHP/Python). On threaded platform, it provides an altogether more scalable and efficient *connection pool*, as described in [this article at ApacheTutor](#). Note that `mod_dbd` supersedes the modules presented in that article.



`mod_dbd` exports five functions for other modules to use. The API is as follows:

```
typedef struct {
    apr_dbd_t *handle;
    apr_dbd_driver_t *driver;
    apr_hash_t *prepared;
} ap_dbd_t;

/* Export functions to access the database */

/* acquire a connection that MUST be explicitly closed.
 * Returns NULL on error
 */
AP_DECLARE(ap_dbd_t*) ap_dbd_open(apr_pool_t*, server_rec*);

/* release a connection acquired with ap_dbd_open */
AP_DECLARE(void) ap_dbd_close(server_rec*, ap_dbd_t*);

/* acquire a connection that will have the lifetime of a request
 * and MUST NOT be explicitly closed. Return NULL on error.
 * This is the preferred function for most applications.
 */
AP_DECLARE(ap_dbd_t*) ap_dbd_acquire(request_rec*);

/* acquire a connection that will have the lifetime of a connection
 * and MUST NOT be explicitly closed. Return NULL on error.
 */
AP_DECLARE(ap_dbd_t*) ap_dbd_cacquire(conn_rec*);

/* Prepare a statement for use by a client module */
AP_DECLARE(void) ap_dbd_prepare(server_rec*, const char*, const char*);

/* Also export them as optional functions for modules that prefer it */
APR_DECLARE_OPTIONAL_FN(ap_dbd_t*, ap_dbd_open, (apr_pool_t*, server_re
APR_DECLARE_OPTIONAL_FN(void, ap_dbd_close, (server_rec*, ap_dbd_t*));
APR_DECLARE_OPTIONAL_FN(ap_dbd_t*, ap_dbd_acquire, (request_rec*));
APR_DECLARE_OPTIONAL_FN(ap_dbd_t*, ap_dbd_cacquire, (conn_rec*));
APR_DECLARE_OPTIONAL_FN(void, ap_dbd_prepare, (server_rec*, const char*
```



`mod_dbd` supports SQL prepared statements on behalf of modules that may wish to use them. Each prepared statement must be assigned a name (label), and they are stored in a hash: the prepared field of an `ap_dbd_t`. Hash entries are of type `apr_dbd_prepared_t` and can be used in any of the `apr_dbd` prepared statement SQL query or select commands.

It is up to `dbd` user modules to use the prepared statements and document what statements can be specified in `httpd.conf`, or to provide their own directives and use `ap_dbd_prepare`.

Caveat

When using prepared statements with a MySQL database, it is preferred to set `reconnect` to 0 in the connection string as to avoid errors that arise from the MySQL client reconnecting without properly resetting the prepared statements. If set to 1, any broken connections will be attempted fixed, but as `mod_dbd` is not informed, the prepared statements will be invalidated.



Any web/database application needs to secure itself against SQL injection attacks. In most cases, Apache DBD is safe, because applications use prepared statements, and untrusted inputs are only ever used as data. Of course, if you use it via third-party modules, you should ascertain what precautions they may require.

However, the *FreeTDS* driver is inherently **unsafe**. The underlying library doesn't support prepared statements, so the driver emulates them, and the untrusted input is merged into the SQL statement.

It can be made safe by *untainting* all inputs: a process inspired by Perl's taint checking. Each input is matched against a regexp, and only the match is used, according to the Perl idiom:

```
$untrusted =~ /([a-z]+)/;  
$trusted = $1;
```

To use this, the untainting regexps must be included in the prepared statements configured. The regexp follows immediately after the % in the prepared statement, and is enclosed in curly brackets {}. For example, if your application expects alphanumeric input, you can use:

```
"SELECT foo FROM bar WHERE input = %s"
```

with other drivers, and suffer nothing worse than a failed query. But with FreeTDS you'd need:

```
"SELECT foo FROM bar WHERE input = % {[A-Za-z0-9]+} s"
```

Now anything that doesn't match the regexp's \$1 match is discarded, so the statement is safe.

An alternative to this may be the third-party ODBC driver, which offers the security of genuine prepared statements.



Description:	Keepalive time for idle connections
Syntax:	DBDExptime <i>time-in-seconds</i>
Default:	DBDExptime 300
Context:	server config, virtual host
Status:	Extension
Module:	mod_dbd

Set the time to keep idle connections alive when the number of connections specified in DBDKeep has been exceeded (threaded platforms only).



Description:	Maximum sustained number of connections
Syntax:	DBDKeep <i>number</i>
Default:	DBDKeep 2
Context:	server config, virtual host
Status:	Extension
Module:	mod_dbd

Set the maximum number of connections per process to be sustained, other than for handling peak demand (threaded platforms only).



Description:	Maximum number of connections
Syntax:	DBDMax <i>number</i>
Default:	DBDMax 10
Context:	server config, virtual host
Status:	Extension
Module:	mod_dbd

Set the hard maximum number of connections per process (threaded platforms only).



Description:	Minimum number of connections
Syntax:	DBDMin <i>number</i>
Default:	DBDMin 1
Context:	server config, virtual host
Status:	Extension
Module:	mod_dbd

Set the minimum number of connections per process (threaded platforms only).



Description:	Parameters for database connection
Syntax:	<code>DBDParams param1=value1[,param2=value2]</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_dbd</code>

As required by the underlying driver. Typically this will be used to pass whatever cannot be defaulted amongst username, password, database name, hostname and port number for connection.

Connection string parameters for current drivers include:

FreeTDS (for MSSQL and SyBase - see SECURITY note)

username, password, appname, dbname, host, charset, lang, server

MySQL

host, port, user, pass, dbname, sock, flags, fldsz, group, reconnect

ODBC

datasource, user, password, connect, ctimeout, stimeout, access, txmode, bufsize

Oracle

user, pass, dbname, server

PostgreSQL

The connection string is passed straight through to `PQconnectdb`

SQLite2

The connection string is split on a colon, and `part1:part2` is used as `sqlite_open(part1, atoi(part2), NULL)`

SQLite3

The connection string is passed straight through to `sqlite3_open`



Description:	Whether to use persistent connections
Syntax:	DBDPersist On Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_dbd

If set to Off, persistent and pooled connections are disabled. A new database connection is opened when requested by a client, and closed immediately on release. This option is for debugging and low-usage servers.

The default is to enable a pool of persistent connections (or a single LAMP-style persistent connection in the case of a non-threaded server), and should almost always be used in operation.

Prior to version 2.2.2, this directive accepted only the values 0 and 1 instead of Off and On, respectively.



Description:	Define an SQL prepared statement
Syntax:	<code>DBDPrepareSQL "SQL statement" label</code>
Context:	server config, virtual host
Status:	Extension
Module:	mod_dbd

For modules such as authentication that repeatedly use a single SQL statement, optimum performance is achieved by preparing the statement at startup rather than every time it is used. This directive prepares an SQL statement and assigns it a label.

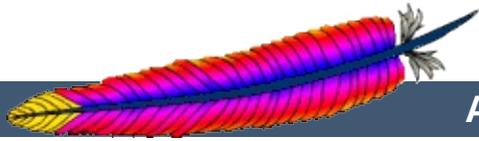


Description:	Specify an SQL driver
Syntax:	<code>DBDriver</code> <i>name</i>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_dbd</code>

Selects an `apr_dbd` driver by name. The driver must be installed on your system (on most systems, it will be a shared object or dll). For example, `DBDriver mysql` will select the MySQL driver in `apr_dbd_mysql.so`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_deflate`

Description:	Compress content before it is delivered to the client
Status:	Extension
Module Identifier:	<code>deflate_module</code>
Source File:	<code>mod_deflate.c</code>

Summary

The `mod_deflate` module provides the DEFLATE output filter that allows output from your server to be compressed before being sent to the client over the network.

See also

[Filters](#)



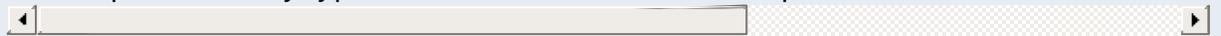
Compression and TLS

Some web applications are vulnerable to an information disclosure attack when a TLS connection carries deflate compressed data. For more information, review the details of the "BREACH" family of attacks.

This is a simple configuration that compresses common text-based content types.

Compress only a few types

```
AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css t
```



Output Compression

Compression is implemented by the DEFLATE [filter](#). The following directive will enable compression for documents in the container where it is placed:

```
SetOutputFilter DEFLATE
```

Some popular browsers cannot handle compression of all content so you may want to set the `gzip-only-text/html` note to `1` to only allow html files to be compressed (see below). If you set this to *anything but 1* it will be ignored.

If you want to restrict the compression to particular MIME types in general, you may use the [AddOutputFilterByType](#) directive. Here is an example of enabling compression only for the html files of the Apache documentation:

```
<Directory "/your-server-root/manual">  
  AddOutputFilterByType DEFLATE text/html  
</Directory>
```

For browsers that have problems even with compression of all file types, use the [BrowserMatch](#) directive to set the `no-gzip` note for that particular browser so that no compression will be performed. You may combine `no-gzip` with `gzip-only-text/html` to get the best results. In that case the former overrides the latter. Take a look at the following excerpt from the [configuration example](#) defined in the section above:

```
BrowserMatch ^Mozilla/4 gzip-only-text/html  
BrowserMatch ^Mozilla/4\.0[678] no-gzip  
BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
```

At first we probe for a User-Agent string that indicates a Netscape

Navigator version of 4.x. These versions cannot handle compression of types other than `text/html`. The versions 4.06, 4.07 and 4.08 also have problems with decompressing html files. Thus, we completely turn off the deflate filter for them.

The third [BrowserMatch](#) directive fixes the guessed identity of the user agent, because the Microsoft Internet Explorer identifies itself also as "Mozilla/4" but is actually able to handle requested compression. Therefore we match against the additional string "MSIE" (`\b` means "word boundary") in the User-Agent Header and turn off the restrictions defined before.

Note

The DEFLATE filter is always inserted after RESOURCE filters like PHP or SSI. It never touches internal subrequests.

Note

There is an environment variable `force-gzip`, set via [SetEnv](#), which will ignore the accept-encoding setting of your browser and will send compressed output.

Output Decompression

The [mod_deflate](#) module also provides a filter for inflating/uncompressing a gzip compressed response body. In order to activate this feature you have to insert the INFLATE filter into the output filter chain using [SetOutputFilter](#) or [AddOutputFilter](#), for example:

```
<Location /dav-area>
  ProxyPass http://example.com/
  SetOutputFilter INFLATE
</Location>
```

This Example will uncompress gzip'ed output from example.com, so other filters can do further processing with it.

Input Decompression

The `mod_deflate` module also provides a filter for decompressing a gzip compressed request body . In order to activate this feature you have to insert the DEFLATE filter into the input filter chain using `SetInputFilter` or `AddInputFilter`, for example:

```
<Location /dav-area>
  SetInputFilter DEFLATE
</Location>
```

Now if a request contains a Content-Encoding: gzip header, the body will be automatically decompressed. Few browsers have the ability to gzip request bodies. However, some special applications actually do support request compression, for instance some [WebDAV](#) clients.

Note on Content-Length

If you evaluate the request body yourself, *don't trust the Content-Length header!* The Content-Length header reflects the length of the incoming data from the client and *not* the byte count of the decompressed data stream.



Learning from proxy servers

The `mod_deflate` module sends a `Vary: Accept-Encoding` HTTP response header to alert proxies that a cached response should be sent only to clients that send the appropriate `Accept-Encoding` request header. This prevents compressed content from being sent to a client that will not understand it.

If you use some special exclusions dependent on, for example, the `User-Agent` header, you must manually configure an addition to the `Vary` header to alert proxies of the additional restrictions. For example, in a typical configuration where the addition of the DEFLATE filter depends on the `User-Agent`, you should add:

```
Header append Vary User-Agent
```

If your decision about compression depends on other information than request headers (e.g. HTTP version), you have to set the `Vary` header to the value `*`. This prevents compliant proxies from caching entirely.

Example

```
Header set Vary *
```



Description:	Fragment size to be compressed at one time by zlib
Syntax:	<code>DeflateBufferSize</code> <i>value</i>
Default:	<code>DeflateBufferSize</code> 8096
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_deflate</code>

The `DeflateBufferSize` directive specifies the size in bytes of the fragments that zlib should compress at one time.



Description:	How much compression do we apply to the output
Syntax:	<code>DeflateCompressionLevel</code> <i>value</i>
Default:	Zlib's default
Context:	server config, virtual host
Status:	Extension
Module:	mod_deflate
Compatibility:	This directive is available since Apache 2.0.45

The `DeflateCompressionLevel` directive specifies what level of compression should be used, the higher the value, the better the compression, but the more CPU time is required to achieve this.

The value must be between 1 (less compression) and 9 (more compression).



Description:	Places the compression ratio in a note for logging
Syntax:	<code>DeflateFilterNote [<i>type</i>] <i>notename</i></code>
Context:	server config, virtual host
Status:	Extension
Module:	mod_deflate
Compatibility:	<i>type</i> is available since Apache 2.0.45

The `DeflateFilterNote` directive specifies that a note about compression ratios should be attached to the request. The name of the note is the value specified for the directive. You can use that note for statistical purposes by adding the value to your [access log](#).

Example

```
DeflateFilterNote ratio

LogFormat '"%r" %b (%{ratio}n) "%{User-agent}i"' deflate
CustomLog logs/deflate_log deflate
```

If you want to extract more accurate values from your logs, you can use the *type* argument to specify the type of data left as a note for logging. *type* can be one of:

Input

Store the byte count of the filter's input stream in the note.

Output

Store the byte count of the filter's output stream in the note.

Ratio

Store the compression ratio ($\text{output}/\text{input} * 100$) in the note. This is the default, if the *type* argument is omitted.

Thus you may log it this way:

Accurate Logging

```
DeflateFilterNote Input instream
DeflateFilterNote Output outstream
DeflateFilterNote Ratio ratio

LogFormat '"%r" %{outstream}n/%{instream}n (%{ratio}n%)' deflate
CustomLog logs/deflate_log deflate
```

See also

- [mod_log_config](#)



Description:	Maximum size of inflated request bodies
Syntax:	<code>DeflateInflateLimitRequestBody</code> <i>value</i>
Default:	None, but <code>LimitRequestBody</code> applies after deflation
Context:	server config, virtual host, directory, <code>.htaccess</code>
Status:	Extension
Module:	<code>mod_deflate</code>
Compatibility:	2.2.28 and later

The `DeflateInflateLimitRequestBody` directive specifies the maximum size of an inflated request body. If it is unset, `LimitRequestBody` is applied to the inflated body.



Description:	Maximum number of times the inflation ratio for request bodies can be crossed
Syntax:	<code>DeflateInflateRatioBurst</code> <i>value</i>
Default:	3
Context:	server config, virtual host, directory, .htaccess
Status:	Extension
Module:	mod_deflate
Compatibility:	2.2.28 and later

The `DeflateInflateRatioBurst` directive specifies the maximum number of times the `DeflateInflateRatioLimit` can be crossed before terminating the request.



Description:	Maximum inflation ratio for request bodies
Syntax:	<code>DeflateInflateRatioLimit</code> <i>value</i>
Default:	200
Context:	server config, virtual host, directory, .htaccess
Status:	Extension
Module:	mod_deflate
Compatibility:	2.2.28 and later

The `DeflateInflateRatioLimit` directive specifies the maximum ratio of deflated to inflated size of an inflated request body. This ratio is checked as the body is streamed in, and if crossed more than `DeflateInflateRatioBurst` times, the request will be terminated.



Description:	How much memory should be used by zlib for compression
Syntax:	<code>DeflateMemLevel</code> <i>value</i>
Default:	<code>DeflateMemLevel 9</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_deflate</code>

The `DeflateMemLevel` directive specifies how much memory should be used by zlib for compression (a value between 1 and 9).

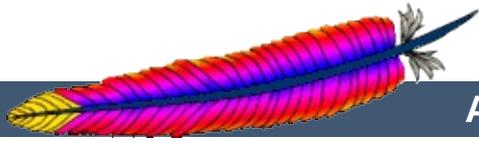


Description:	Zlib compression window size
Syntax:	<code>DeflateWindowSize</code> <i>value</i>
Default:	<code>DeflateWindowSize</code> 15
Context:	server config, virtual host
Status:	Extension
Module:	mod_deflate

The `DeflateWindowSize` directive specifies the zlib compression window size (a value between 1 and 15). Generally, the higher the window size, the higher can the compression ratio be expected.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_dir`

Description:	Provides for "trailing slash" redirects and serving directory index files
Status:	Base
Module Identifier:	<code>dir_module</code>
Source File:	<code>mod_dir.c</code>

Summary

The index of a directory can come from one of two sources:

- A file written by the user, typically called `index.html`. The `DirectoryIndex` directive sets the name of this file. This is controlled by `mod_dir`.
- Otherwise, a listing generated by the server. This is provided by `mod_autoindex`.

The two functions are separated so that you can completely remove (or replace) automatic index generation should you want to.

A "trailing slash" redirect is issued when the server receives a request for a URL `http://servername/foo/dirname` where `dirname` is a directory. Directories require a trailing slash, so `mod_dir` issues a redirect to `http://servername/foo/dirname/`.



Description:	List of resources to look for when the client requests a directory
Syntax:	<code>DirectoryIndex <i>local-url</i> [<i>local-url</i>] ...</code>
Default:	<code>DirectoryIndex index.html</code>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_dir

The `DirectoryIndex` directive sets the list of resources to look for, when the client requests an index of the directory by specifying a `/` at the end of the directory name. *Local-url* is the (%-encoded) URL of a document on the server relative to the requested directory; it is usually the name of a file in the directory. Several URLs may be given, in which case the server will return the first one that it finds. If none of the resources exist and the `Indexes` option is set, the server will generate its own listing of the directory.

Example

```
DirectoryIndex index.html
```

then a request for `http://example.com/docs/` would return `http://example.com/docs/index.html` if it exists, or would list the directory if it did not.

Note that the documents do not need to be relative to the directory;

```
DirectoryIndex index.html index.txt /cgi-bin/index.pl
```

would cause the CGI script `/cgi-bin/index.pl` to be executed if neither `index.html` or `index.txt` existed in a directory.

Note: Multiple `DirectoryIndex` directives within the same context will add to the list of resources to look for rather than replace:

```
# Example A: Set index.html as an index page, then add index.php to the list
<Directory /foo>
    DirectoryIndex index.html
    DirectoryIndex index.php
</Directory>
```

```
# Example B: This is identical to example A, except it's done with a single directive
<Directory /foo>
    DirectoryIndex index.html index.php
</Directory>
```



Description:	Toggle trailing slash redirects on or off
Syntax:	DirectorySlash On Off
Default:	DirectorySlash On
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_dir
Compatibility:	Available in version 2.0.51 and later

The `DirectorySlash` directive determines whether `mod_dir` should fixup URLs pointing to a directory or not.

Typically if a user requests a resource without a trailing slash, which points to a directory, `mod_dir` redirects him to the same resource, but *with* trailing slash for some good reasons:

- The user is finally requesting the canonical URL of the resource
- `mod_autoindex` works correctly. Since it doesn't emit the path in the link, it would point to the wrong path.
- `DirectoryIndex` will be evaluated *only* for directories requested with trailing slash.
- Relative URL references inside html pages will work correctly.

If you don't want this effect *and* the reasons above don't apply to you, you can turn off the redirect as shown below. However, be aware that there are possible security implications to doing this.

```
# see security warning below!  
<Location /some/path>  
    DirectorySlash Off  
    SetHandler some-handler  
</Location>
```

Security Warning

Turning off the trailing slash redirect may result in an information disclosure. Consider a situation where `mod_autoindex` is active (`Options +Indexes`) and `DirectoryIndex` is set to a valid resource (say, `index.html`) and there's no other special handler defined for that URL. In this case a request with a trailing slash would show the `index.html` file. **But a request without trailing slash would list the directory contents.**



Description:	Define a default URL for requests that don't map to a file
Syntax:	<code>FallbackResource disabled local-url</code>
Default:	None - httpd will return 404 (Not Found)
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_dir
Compatibility:	Apache HTTP Server 2.2.16 and later - The disabled argument is supported since 2.2.24

Use this to set a handler for any URL that doesn't map to anything in your filesystem, and would otherwise return HTTP 404 (Not Found). For example

```
FallbackResource /not-404.php
```

will cause requests for non-existent files to be handled by `not-404.php`, while requests for files that exist are unaffected.

It is frequently desirable to have a single file or resource handle all requests to a particular directory, except those requests that correspond to an existing file or script. This is often referred to as a 'front controller.'

In earlier versions of httpd, this effect typically required `mod_rewrite`, and the use of the `-f` and `-d` tests for file and directory existence. This now requires only one line of configuration.

```
FallbackResource /index.php
```

Existing files, such as images, css files, and so on, will be served normally.

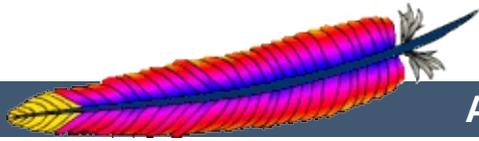
Use the `disabled` argument to disable that feature if inheritance from a parent directory is not desired.

In a sub-URI, such as *http://example.com/blog/* this *sub-URI* has to be supplied as *local-url*:

```
<Directory /web/example.com/htdocs/blog>
  FallbackResource /blog/index.php
</Directory>
<Directory /web/example.com/htdocs/blog/images>
  FallbackResource disabled
</Directory>
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_disk_cache`

Description:	Content cache storage manager keyed to URIs
Status:	Extension
Module Identifier:	<code>disk_cache_module</code>
Source File:	<code>mod_disk_cache.c</code>

Summary

`mod_disk_cache` implements a disk based storage manager. It is primarily of use in conjunction with `mod_cache`.

Content is stored in and retrieved from the cache using URI based keys. Content with access protection is not cached.

`htcacheclean` can be used to maintain the cache size at a maximum level.

Note:

`mod_disk_cache` requires the services of `mod_cache`.

Note:

`mod_disk_cache` uses the `sendfile` feature to serve files from the cache when supported by the platform, and when enabled with `EnableSendfile`. However, per-directory and `.htaccess` configuration of `EnableSendfile` are ignored by `mod_disk_cache` as the corresponding settings are not available to the module when a request is being served from the cache.



Description:	The number of characters in subdirectory names
Syntax:	CacheDirLength <i>length</i>
Default:	CacheDirLength 2
Context:	server config, virtual host
Status:	Extension
Module:	mod_disk_cache

The `CacheDirLength` directive sets the number of characters for each subdirectory name in the cache hierarchy.

The result of `CacheDirLevels`* `CacheDirLength` must not be higher than 20.

CacheDirLength 4



Description:	The number of levels of subdirectories in the cache.
Syntax:	CacheDirLevels <i>levels</i>
Default:	CacheDirLevels 3
Context:	server config, virtual host
Status:	Extension
Module:	mod_disk_cache

The `CacheDirLevels` directive sets the number of subdirectory levels in the cache. Cached data will be saved this many directory levels below the `CacheRoot` directory.

The result of `CacheDirLevels* CacheDirLength` must not be higher than 20.

```
CacheDirLevels 5
```



Description:	The maximum size (in bytes) of a document to be placed in the cache
Syntax:	CacheMaxFileSize <i>bytes</i>
Default:	CacheMaxFileSize 1000000
Context:	server config, virtual host
Status:	Extension
Module:	mod_disk_cache

The `CacheMaxFileSize` directive sets the maximum size, in bytes, for a document to be considered for storage in the cache.

```
CacheMaxFileSize 64000
```



Description:	The minimum size (in bytes) of a document to be placed in the cache
Syntax:	CacheMinFileSize <i>bytes</i>
Default:	CacheMinFileSize 1
Context:	server config, virtual host
Status:	Extension
Module:	mod_disk_cache

The `CacheMinFileSize` directive sets the minimum size, in bytes, for a document to be considered for storage in the cache.

```
CacheMinFileSize 64
```



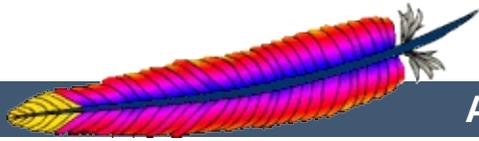
Description:	The directory root under which cache files are stored
Syntax:	CacheRoot <i>directory</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_disk_cache

The `CacheRoot` directive defines the name of the directory on the disk to contain cache files. If the `mod_disk_cache` module has been loaded or compiled in to the Apache server, this directive *must* be defined. Failing to provide a value for `CacheRoot` will result in a configuration file processing error. The `CacheDirLevels` and `CacheDirLength` directives define the structure of the directories under the specified root directory.

```
CacheRoot c:/cacheroot
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_dumpio

Description:	Dumps all I/O to error log as desired.
Status:	Extension
Module Identifier:	dumpio_module
Source File:	mod_dumpio.c

Summary

mod_dumpio allows for the logging of all input received by Apache and/or all output sent by Apache to be logged (dumped) to the error.log file.

The data logging is done right after SSL decoding (for input) and right before SSL encoding (for output). As can be expected, this can produce extreme volumes of data, and should only be used when debugging problems.



Enabling module support

To enable the module, it should be compiled and loaded in to your running Apache configuration. Logging can then be enabled or disabled via the below directives.



Description:	Dump all input data to the error log
Syntax:	DumpIOInput On Off
Default:	DumpIOInput Off
Context:	server config
Status:	Extension
Module:	mod_dumpio
Compatibility:	DumpIOInput is only available in Apache 2.1.3 and later.

Enable dumping of all input.

Example

```
DumpIOInput On
```



DumpIOLogLevel Directive

Description:	Controls the logging level of the DumpIO output
Syntax:	DumpIOLogLevel <i>level</i>
Default:	DumpIOLogLevel debug
Context:	server config
Status:	Extension
Module:	mod_dumpio
Compatibility:	DumpIOLogLevel is only available in Apache 2.2.4 and later.

Enable dumping of all output at a specific [LogLevel](#) level.

Example

```
DumpIOLogLevel notice
```

Compatibility

Prior to 2.2.4 [mod_dumpio](#) would only dump to the log when [LogLevel](#) was set to debug



Description:	Dump all output data to the error log
Syntax:	DumpIOOutput On Off
Default:	DumpIOOutput Off
Context:	server config
Status:	Extension
Module:	mod_dumpio
Compatibility:	DumpIOOutput is only available in Apache 2.1.3 and later.

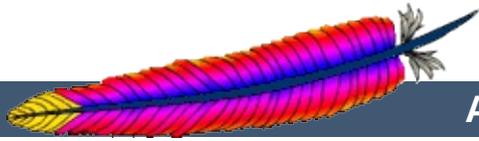
Enable dumping of all output.

Example

```
DumpIOOutput On
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_echo

Description:	A simple echo server to illustrate protocol modules
Status:	Experimental
Module Identifier:	echo_module
Source File:	mod_echo.c
Compatibility:	Available in Apache 2.0 and later

Summary

This module provides an example protocol module to illustrate the concept. It provides a simple echo server. Telnet to it and type stuff, and it will echo it.



Description:	Turn the echo server on or off
Syntax:	ProtocolEcho On Off
Default:	ProtocolEcho Off
Context:	server config, virtual host
Status:	Experimental
Module:	mod_echo
Compatibility:	ProtocolEcho is only available in 2.0 and later.

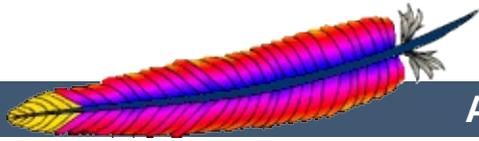
The **ProtocolEcho** directive enables or disables the echo server.

Example

```
ProtocolEcho On
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_env

Description:	Modifies the environment which is passed to CGI scripts and SSI pages
Status:	Base
Module Identifier:	env_module
Source File:	mod_env.c

Summary

This module allows for control of internal environment variables that are used by various Apache HTTP Server modules. These variables are also provided to CGI scripts as native system environment variables, and available for use in SSI pages. Environment variables may be passed from the shell which invoked the [httpd](#) process. Alternatively, environment variables may be set or unset within the configuration process.

See also

[Environment Variables](#)

[SetEnvIf](#)



Description:	Passes environment variables from the shell
Syntax:	<code>PassEnv <i>env-variable</i> [<i>env-variable</i>] ...</code>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_env

Specifies one or more native system environment variables to make available as internal environment variables, which are available to Apache HTTP Server modules as well as propagated to CGI scripts and SSI pages. Values come from the native OS environment of the shell which invoked the [httpd](#) process.

Example

```
PassEnv LD_LIBRARY_PATH
```



Description:	Sets environment variables
Syntax:	<code>SetEnv <i>env-variable</i> [<i>value</i>]</code>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_env

Sets an internal environment variable, which is then available to Apache HTTP Server modules, and passed on to CGI scripts and SSI pages.

Example

```
SetEnv SPECIAL_PATH /foo/bin
```

If you omit the *value* argument, the variable is set to an empty string.

The internal environment variables set by this directive are set *after* most early request processing directives are run, such as access control and URI-to-filename mapping. If the environment variable you're setting is meant as input into this early phase of processing such as the [RewriteRule](#) directive, you should instead set the environment variable with [SetEnvIf](#).

On 2.2, the PATH environment variable cannot be set using Setenv.

See also

- [Environment Variables](#)



Description:	Removes variables from the environment
Syntax:	<code>UnsetEnv <i>env-variable</i> [<i>env-variable</i>] ...</code>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_env

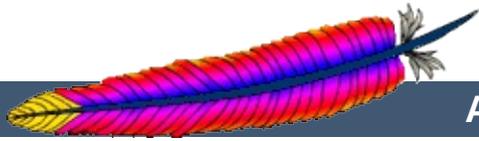
Removes one or more internal environment variables from those passed on to CGI scripts and SSI pages.

Example

```
UnsetEnv LD_LIBRARY_PATH
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_example`

Description:	Illustrates the Apache module API
Status:	Experimental
Module Identifier:	<code>example_module</code>
Source File:	<code>mod_example.c</code>

Summary

Some files in the `modules/experimental` directory under the Apache distribution directory tree are provided as an example to those that wish to write modules that use the Apache API.

The main file is `mod_example.c`, which illustrates all the different callback mechanisms and call syntaxes. By no means does an add-on module need to include routines for all of the callbacks - quite the contrary!

The example module is an actual working module. If you link it into your server, enable the "example-handler" handler for a location, and then browse to that location, you will see a display of some of the tracing the example module did as the various callbacks were made.



Configuring the Example Module

To include the example module in your server, follow the steps below:

1. Run `configure` with `--enable-example` option.
2. Make the server (run "make").

To add another module of your own:

- A. `cp modules/experimental/mod_example.c modules/new_module/mod_myexample.c`
- B. Modify the file.
- C. Create `modules/new_module/config.m4`.
 1. Add `APACHE_MODPATH_INIT(new_module)`.
 2. Copy `APACHE_MODULE` line with "example" from `modules/experimental/config.m4`.
 3. Replace the first argument "example" with *myexample*.
 4. Replace the second argument with brief description of your module. It will be used in `configure --help`.
 5. If your module needs additional C compiler flags, linker flags or libraries, add them to `CFLAGS`, `LDFLAGS` and `LIBS` accordingly. See other `config.m4` files in `modules` directory for examples.
 6. Add `APACHE_MODPATH_FINISH`.
- D. Create `module/new_module/Makefile.in`. If your module doesn't need special build instructions, all you need to have in that file is `include $(top_srcdir)/build/special.mk`.
- E. Run `./buildconf` from the top-level directory.
- F. Build the server with `--enable-myexample`



To activate the example module, include a block similar to the following in your `httpd.conf` file:

```
<Location /example-info>  
SetHandler example-handler  
</Location>
```

As an alternative, you can put the following into a `.htaccess` file and then request the file "test.example" from that location:

```
AddHandler example-handler .example
```

After reloading/restarting your server, you should be able to browse to this location and see the brief display mentioned earlier.

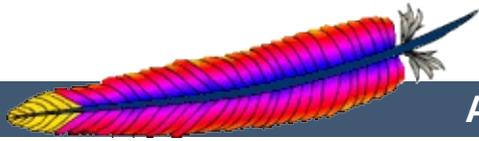


Description:	Demonstration directive to illustrate the Apache module API
Syntax:	Example
Context:	server config, virtual host, directory, .htaccess
Status:	Experimental
Module:	mod_example

The `Example` directive just sets a demonstration flag which the example module's content handler displays. It takes no arguments. If you browse to an URL to which the example content-handler applies, you will get a display of the routines within the module and how and in what order they were called to service the document request. The effect of this directive one can observe under the point "Example directive declared here: YES/NO".

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_expires

Description:	Generation of Expires and Cache-Control HTTP headers according to user-specified criteria
Status:	Extension
Module Identifier:	expires_module
Source File:	mod_expires.c

Summary

This module controls the setting of the Expires HTTP header and the max-age directive of the Cache-Control HTTP header in server responses. The expiration date can set to be relative to either the time the source file was last modified, or to the time of the client access.

These HTTP headers are an instruction to the client about the document's validity and persistence. If cached, the document may be fetched from the cache rather than from the source until this time has passed. After that, the cache copy is considered "expired" and invalid, and a new copy must be obtained from the source.

To modify Cache-Control directives other than max-age (see [RFC 2616 section 14.9](#)), you can use the [Header](#) directive.

When the Expires header is already part of the response generated by the server, for example when generated by a CGI script or proxied from an origin server, this module does not change or add an Expires or Cache-Control header.



The `ExpiresDefault` and `ExpiresByType` directives can also be defined in a more readable syntax of the form:

```
ExpiresDefault "base [plus num type] [num type] ..."  
ExpiresByType type/encoding "base [plus num type] [num type] ..."
```

where *base* is one of:

- `access`
- `now` (equivalent to 'access')
- `modification`

The `plus` keyword is optional. *num* should be an integer value [acceptable to `atoi()`], and *type* is one of:

- `years`
- `months`
- `weeks`
- `days`
- `hours`
- `minutes`
- `seconds`

For example, any of the following directives can be used to make documents expire 1 month after being accessed, by default:

```
ExpiresDefault "access plus 1 month"  
ExpiresDefault "access plus 4 weeks"  
ExpiresDefault "access plus 30 days"
```

The expiry time can be fine-tuned by adding several '*num type*' clauses:

```
ExpiresByType text/html "access plus 1 month 15 days 2 hours"  
ExpiresByType image/gif "modification plus 5 hours 3 minutes"
```

Note that if you use a modification date based setting, the Expires header will **not** be added to content that does not come from a file on disk. This is due to the fact that there is no modification time for such content.



Description:	Enables generation of Expires headers
Syntax:	ExpiresActive On Off
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Extension
Module:	mod_expires

This directive enables or disables the generation of the Expires and Cache-Control headers for the document realm in question. (That is, if found in an .htaccess file, for instance, it applies only to documents generated from that directory.) If set to Off, the headers will not be generated for any document in the realm (unless overridden at a lower level, such as an .htaccess file overriding a server config file). If set to On, the headers will be added to served documents according to the criteria defined by the [ExpiresByType](#) and [ExpiresDefault](#) directives (*q.v.*).

Note that this directive does not guarantee that an Expires or Cache-Control header will be generated. If the criteria aren't met, no header will be sent, and the effect will be as though this directive wasn't even specified.



ExpiresByType Directive

Description:	Value of the Expires header configured by MIME type
Syntax:	ExpiresByType <i>MIME-type</i> <i><code></i> seconds
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Extension
Module:	mod_expires

This directive defines the value of the Expires header and the max-age directive of the Cache-Control header generated for documents of the specified type (e.g., text/html). The second argument sets the number of seconds that will be added to a base time to construct the expiration date. The Cache-Control: max-age is calculated by subtracting the request time from the expiration date and expressing the result in seconds.

The base time is either the last modification time of the file, or the time of the client's access to the document. Which should be used is specified by the *<code>* field; M means that the file's last modification time should be used as the base time, and A means the client's access time should be used.

The difference in effect is subtle. If M is used, all current copies of the document in all caches will expire at the same time, which can be good for something like a weekly notice that's always found at the same URL. If A is used, the date of expiration is different for each client; this can be good for image files that don't change very often, particularly for a set of related documents that all refer to the same images (i.e., the images will be accessed repeatedly within a relatively short timespan).

Example:

```
# enable expirations
ExpiresActive On
# expire GIF images after a month in the client's cache
```

```
ExpiresByType image/gif A2592000
# HTML documents are good for a week from the
# time they were changed
ExpiresByType text/html M604800
```

Note that this directive only has effect if `ExpiresActive On` has been specified. It overrides, for the specified MIME type *only*, any expiration date set by the [ExpiresDefault](#) directive.

You can also specify the expiration time calculation using an [alternate syntax](#), described earlier in this document.

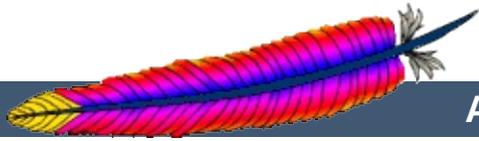


Description:	Default algorithm for calculating expiration time
Syntax:	ExpiresDefault <i><code>seconds</i>
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Extension
Module:	mod_expires

This directive sets the default algorithm for calculating the expiration time for all documents in the affected realm. It can be overridden on a type-by-type basis by the [ExpiresByType](#) directive. See the description of that directive for details about the syntax of the argument, and the [alternate syntax](#) description as well.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_ext_filter`

Description:	Pass the response body through an external program before delivery to the client
Status:	Extension
Module Identifier:	<code>ext_filter_module</code>
Source File:	<code>mod_ext_filter.c</code>

Summary

`mod_ext_filter` presents a simple and familiar programming model for [filters](#). With this module, a program which reads from stdin and writes to stdout (i.e., a Unix-style filter command) can be a filter for Apache. This filtering mechanism is much slower than using a filter which is specially written for the Apache API and runs inside of the Apache server process, but it does have the following benefits:

- the programming model is much simpler
- any programming/scripting language can be used, provided that it allows the program to read from standard input and write to standard output
- existing programs can be used unmodified as Apache filters

Even when the performance characteristics are not suitable for production use, `mod_ext_filter` can be used as a prototype environment for filters.

See also

[Filters](#)



Generating HTML from some other type of response

```
# mod_ext_filter directive to define a filter
# to HTML-ize text/c files using the external
# program /usr/bin/enscript, with the type of
# the result set to text/html
ExtFilterDefine c-to-html mode=output \
    intype=text/c outtype=text/html \
    cmd="/usr/bin/enscript --color -W html -Ec -o - -"

<Directory "/export/home/trawick/apacheinst/htdocs/c">
    # core directive to cause the new filter to
    # be run on output
    SetOutputFilter c-to-html

    # mod_mime directive to set the type of .c
    # files to text/c
    AddType text/c .c

    # mod_ext_filter directive to set the debug
    # level just high enough to see a log message
    # per request showing the configuration in force
    ExtFilterOptions DebugLevel=1
</Directory>
```

Implementing a content encoding filter

Note: this gzip example is just for the purposes of illustration. Please refer to [mod_deflate](#) for a practical implementation.

```
# mod_ext_filter directive to define the external filter
ExtFilterDefine gzip mode=output cmd=/bin/gzip

<Location /gzipped>
    # core directive to cause the gzip filter to be
    # run on output
    SetOutputFilter gzip

    # mod_header directive to add
    # "Content-Encoding: gzip" header field
    Header set Content-Encoding gzip
</Location>
```

Slowing down the server

```
# mod_ext_filter directive to define a filter
# which runs everything through cat; cat doesn't
# modify anything; it just introduces extra pathlength
# and consumes more resources
ExtFilterDefine slowdown mode=output cmd=/bin/cat \
    preservescontentlength

<Location />
    # core directive to cause the slowdown filter to
    # be run several times on output
    #
    SetOutputFilter slowdown;slowdown;slowdown
</Location>
```

Using sed to replace text in the response

```
# mod_ext_filter directive to define a filter which
# replaces text in the response
#
ExtFilterDefine fixtext mode=output intype=text/html \
    cmd="/bin/sed s/verdana/arial/g"

<Location />
    # core directive to cause the fixtext filter to
    # be run on output
    SetOutputFilter fixtext
</Location>
```

You can do the same thing using [mod_substitute](#) without invoking an external process.

Tracing another filter

```
# Trace the data read and written by mod_deflate
# for a particular client (IP 192.168.1.31)
# experiencing compression problems.
# This filter will trace what goes into mod_deflate.
ExtFilterDefine tracebefore \
    cmd="/bin/tracefilter.pl /tmp/tracebefore" \
```

```
EnableEnv=trace_this_client

# This filter will trace what goes after mod_deflate.
# Note that without the ftype parameter, the default
# filter type of AP_FTYPE_RESOURCE would cause the
# filter to be placed *before* mod_deflate in the filter
# chain. Giving it a numeric value slightly higher than
# AP_FTYPE_CONTENT_SET will ensure that it is placed
# after mod_deflate.
ExtFilterDefine traceafter \
    cmd="/bin/tracefilter.pl /tmp/traceafter" \
    EnableEnv=trace_this_client ftype=21

<Directory /usr/local/docs>
    SetEnvIf Remote_Addr 192.168.1.31 trace_this_client
    SetOutputFilter tracebefore;deflate;traceafter
</Directory>
```

Here is the filter which traces the data:

```
#!/usr/local/bin/perl -w
use strict;

open(SAVE, ">$ARGV[0]")
    or die "can't open $ARGV[0]: $?";

while (<STDIN>) {
    print SAVE $_;
    print $_;
}

close(SAVE);
```



Description:	Define an external filter
Syntax:	<code>ExtFilterDefine <i>filtername</i> <i>parameters</i></code>
Context:	server config
Status:	Extension
Module:	<code>mod_ext_filter</code>

The `ExtFilterDefine` directive defines the characteristics of an external filter, including the program to run and its arguments.

filtername specifies the name of the filter being defined. This name can then be used in `SetOutputFilter` directives. It must be unique among all registered filters. *At the present time, no error is reported by the register-filter API, so a problem with duplicate names isn't reported to the user.*

Subsequent parameters can appear in any order and define the external command to run and certain other characteristics. The only required parameter is `cmd=`. These parameters are:

`cmd=cmdline`

The `cmd=` keyword allows you to specify the external command to run. If there are arguments after the program name, the command line should be surrounded in quotation marks (*e.g.*, `cmd="/bin/mypgm arg1 arg2"`.) Normal shell quoting is not necessary since the program is run directly, bypassing the shell. Program arguments are blank-delimited. A backslash can be used to escape blanks which should be part of a program argument. Any backslashes which are part of the argument must be escaped with backslash themselves. In addition to the standard CGI environment variables, `DOCUMENT_URI`, `DOCUMENT_PATH_INFO`, and `QUERY_STRING_UNESCAPED` will also be set for the program.

`mode=mode`

Use `mode=output` (the default) for filters which process the response. Use `mode=input` for filters which process the request. `mode=input` is available in Apache 2.1 and later.

`intype=imt`

This parameter specifies the internet media type (*i.e.*, MIME type) of documents which should be filtered. By default, all documents are filtered. If `intype=` is specified, the filter will be disabled for documents of other types.

`outtype=imt`

This parameter specifies the internet media type (*i.e.*, MIME type) of filtered documents. It is useful when the filter changes the internet media type as part of the filtering operation. By default, the internet media type is unchanged.

`PreservesContentLength`

The `PreservesContentLength` keyword specifies that the filter preserves the content length. This is not the default, as most filters change the content length. In the event that the filter doesn't modify the length, this keyword should be specified.

`fctype=filtertype`

This parameter specifies the numeric value for filter type that the filter should be registered as. The default value, `AP_FCTYPE_RESOURCE`, is sufficient in most cases. If the filter needs to operate at a different point in the filter chain than resource filters, then this parameter will be necessary. See the `AP_FCTYPE_foo` definitions in `util_filter.h` for appropriate values.

`disableenv=env`

This parameter specifies the name of an environment variable which, if set, will disable the filter.

`enableenv=env`

This parameter specifies the name of an environment variable which must be set, or the filter will be disabled.



Description:	Configure <code>mod_ext_filter</code> options
Syntax:	<code>ExtFilterOptions option [option] ...</code>
Default:	<code>ExtFilterOptions DebugLevel=0 NoLogStderr</code>
Context:	directory
Status:	Extension
Module:	<code>mod_ext_filter</code>

The `ExtFilterOptions` directive specifies special processing options for `mod_ext_filter`. *Option* can be one of

DebugLevel=*n*

The `DebugLevel` keyword allows you to specify the level of debug messages generated by `mod_ext_filter`. By default, no debug messages are generated. This is equivalent to `DebugLevel=0`. With higher numbers, more debug messages are generated, and server performance will be degraded. The actual meanings of the numeric values are described with the definitions of the `DBGLVL_` constants near the beginning of `mod_ext_filter.c`.

Note: The core directive `LogLevel` should be used to cause debug messages to be stored in the Apache error log.

LogStderr | NoLogStderr

The `LogStderr` keyword specifies that messages written to standard error by the external filter program will be saved in the Apache error log. `NoLogStderr` disables this feature.

Onfail=[abort | remove] (new in httpd version 2.2.12).

Determines how to proceed if the external filter program cannot be started. With `abort` (the default value) the request will be aborted. With `remove`, the filter is removed and the request continues without it.

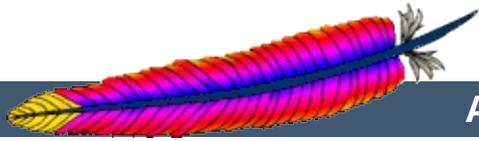
Example

```
ExtFilterOptions LogStderr DebugLevel=0
```

Messages written to the filter's standard error will be stored in the Apache error log. No debug messages will be generated by [mod_ext_filter](#).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_file_cache`

Description:	Caches a static list of files in memory
Status:	Experimental
Module Identifier:	<code>file_cache_module</code>
Source File:	<code>mod_file_cache.c</code>

Summary

This module should be used with care. You can easily create a broken site using `mod_file_cache`, so read this document carefully.

Caching frequently requested files that change very infrequently is a technique for reducing server load. `mod_file_cache` provides two techniques for caching frequently requested *static* files. Through configuration directives, you can direct `mod_file_cache` to either open then `mmap()` a file, or to pre-open a file and save the file's open *file handle*. Both techniques reduce server load when processing requests for these files by doing part of the work (specifically, the file I/O) for serving the file when the server is started rather than during each request.

Notice: You cannot use this for speeding up CGI programs or other files which are served by special content handlers. It can only be used for regular files which are usually served by the Apache core content handler.

This module is an extension of and borrows heavily from the `mod_mmap_static` module in Apache 1.3.



`mod_file_cache` caches a list of statically configured files via `MMapFile` or `CacheFile` directives in the main server configuration.

Not all platforms support both directives. You will receive an error message in the server error log if you attempt to use an unsupported directive. If given an unsupported directive, the server will start but the file will not be cached. On platforms that support both directives, you should experiment with both to see which works best for you.

MMapFile Directive

The `MMapFile` directive of `mod_file_cache` maps a list of statically configured files into memory through the system call `mmap()`. This system call is available on most modern Unix derivatives, but not on all. There are sometimes system-specific limits on the size and number of files that can be `mmap()`ed, experimentation is probably the easiest way to find out.

This `mmap()`ing is done once at server start or restart, only. So whenever one of the mapped files changes on the filesystem you *have* to restart the server (see the [Stopping and Restarting](#) documentation). To reiterate that point: if the files are modified *in place* without restarting the server you may end up serving requests that are completely bogus. You should update files by unlinking the old copy and putting a new copy in place. Most tools such as `rdist` and `mv` do this. The reason why this module doesn't take care of changes to the files is that this check would need an extra `stat()` every time which is a waste and against the intent of I/O reduction.

CacheFile Directive

The `CacheFile` directive of `mod_file_cache` opens an active *handle* or *file descriptor* to the file (or files) listed in the configuration directive and places these open file handles in the cache. When the file is

requested, the server retrieves the handle from the cache and passes it to the `sendfile()` (or `TransmitFile()` on Windows), socket API.

This file handle caching is done once at server start or restart, only. So whenever one of the cached files changes on the filesystem you *have* to restart the server (see the [Stopping and Restarting](#) documentation). To reiterate that point: if the files are modified *in place* without restarting the server you may end up serving requests that are completely bogus. You should update files by unlinking the old copy and putting a new copy in place. Most tools such as `rdist` and `mv` do this.

Note

Don't bother asking for a directive which recursively caches all the files in a directory. Try this instead... See the [Include](#) directive, and consider this command:

```
find /www/htdocs -type f -print \  
| sed -e 's/./mmapfile &/' > /www/conf/mmap.conf
```



Description:	Cache a list of file handles at startup time
Syntax:	CacheFile <i>file-path</i> [<i>file-path</i>] ...
Context:	server config
Status:	Experimental
Module:	mod_file_cache

The `CacheFile` directive opens handles to one or more files (given as whitespace separated arguments) and places these handles into the cache at server startup time. Handles to cached files are automatically closed on a server shutdown. When the files have changed on the filesystem, the server should be restarted to re-cache them.

Be careful with the *file-path* arguments: They have to literally match the filesystem path Apache's URL-to-filename translation handlers create. We cannot compare inodes or other stuff to match paths through symbolic links *etc.* because that again would cost extra `stat()` system calls which is not acceptable. This module may or may not work with filenames rewritten by `mod_alias` or `mod_rewrite`.

Example

```
CacheFile /usr/local/apache/htdocs/index.html
```



Description:	Map a list of files into memory at startup time
Syntax:	MMapFile <i>file-path</i> [<i>file-path</i>] ...
Context:	server config
Status:	Experimental
Module:	mod_file_cache

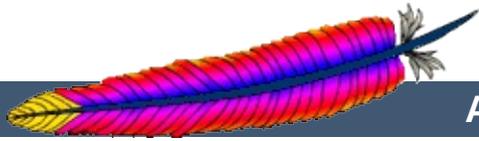
The **MMapFile** directive maps one or more files (given as whitespace separated arguments) into memory at server startup time. They are automatically unmapped on a server shutdown. When the files have changed on the filesystem at least a HUP or USR1 signal should be send to the server to re-mmap() them.

Be careful with the *file-path* arguments: They have to literally match the filesystem path Apache's URL-to-filename translation handlers create. We cannot compare inodes or other stuff to match paths through symbolic links *etc.* because that again would cost extra `stat()` system calls which is not acceptable. This module may or may not work with filenames rewritten by [mod_alias](#) or [mod_rewrite](#).

Example

```
MMapFile /usr/local/apache/htdocs/index.html
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_filter`

Description:	Context-sensitive smart filter configuration module
Status:	Base
Module Identifier:	<code>filter_module</code>
Source File:	<code>mod_filter.c</code>
Compatibility:	Version 2.1 and later

Summary

This module enables smart, context-sensitive configuration of output content filters. For example, apache can be configured to process different content-types through different filters, even when the content-type is not known in advance (e.g. in a proxy).

`mod_filter` works by introducing indirection into the filter chain. Instead of inserting filters in the chain, we insert a filter harness which in turn dispatches conditionally to a filter provider. Any content filter may be used as a provider to `mod_filter`; no change to existing filter modules is required (although it may be possible to simplify them).



Content Filtering

In the traditional filtering model, filters are inserted unconditionally using [AddOutputFilter](#) and family. Each filter then needs to determine whether to run, and there is little flexibility available for server admins to allow the chain to be configured dynamically.

[mod_filter](#) by contrast gives server administrators a great deal of flexibility in configuring the filter chain. In fact, filters can be inserted based on any Request Header, Response Header or Environment Variable. This generalises the limited flexibility offered by [AddOutputFilterByType](#), and fixes it to work correctly with dynamic content, regardless of the content generator. The ability to dispatch based on Environment Variables offers the full flexibility of configuration with [mod_rewrite](#) to anyone who needs it.



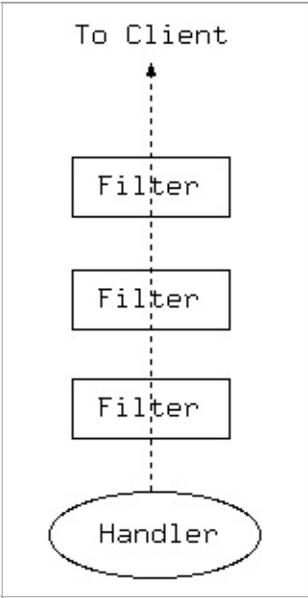


Figure 1: The traditional filter model

In the traditional model, output filters are a simple chain from the content generator (handler) to the client. This works well provided the filter chain can be correctly configured, but presents problems when the filters need to be configured dynamically based on the outcome of the handler.

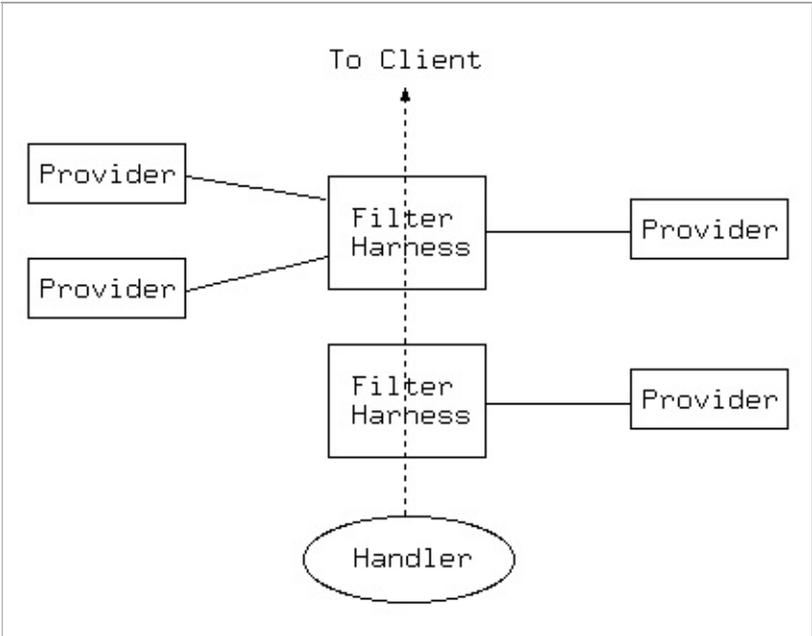


Figure 2: The mod filter model

mod filter works by introducing indirection into the filter chain. Instead

of inserting filters in the chain, we insert a filter harness which in turn dispatches conditionally to a filter provider. Any content filter may be used as a provider to [mod_filter](#); no change to existing filter modules is required (although it may be possible to simplify them). There can be multiple providers for one filter, but no more than one provider will run for any single request.

A filter chain comprises any number of instances of the filter harness, each of which may have any number of providers. A special case is that of a single provider with unconditional dispatch: this is equivalent to inserting the provider filter directly into the chain.



There are three stages to configuring a filter chain with `mod_filter`. For details of the directives, see below.

Declare Filters

The `FilterDeclare` directive declares a filter, assigning it a name and filter type. Required only if the filter is not the default type `AP_FTYPE_RESOURCE`.

Register Providers

The `FilterProvider` directive registers a provider with a filter. The filter may have been declared with `FilterDeclare`; if not, `FilterProvider` will implicitly declare it with the default type `AP_FTYPE_RESOURCE`. The provider must have been registered with `ap_register_output_filter` by some module. The remaining arguments to `FilterProvider` are a dispatch criterion and a match string. The former may be an HTTP request or response header, an environment variable, or the Handler used by this request. The latter is matched to it for each request, to determine whether this provider will be used to implement the filter for this request.

Configure the Chain

The above directives build components of a smart filter chain, but do not configure it to run. The `FilterChain` directive builds a filter chain from smart filters declared, offering the flexibility to insert filters at the beginning or end of the chain, remove a filter, or clear the chain.



Filtering and response status

`mod_filter` normally only runs filters on responses with HTTP status 200 (OK). If you want to filter documents with other response statuses, you can set the `filter-errordocs` environment variable, and it will work on all responses regardless of status. To refine this further, you can use expression conditions with `FilterProvider`.



Server side Includes (SSI)

A simple case of using `mod_filter` in place of `AddOutputFilterByType`

```
FilterDeclare SSI
FilterProvider SSI INCLUDES resp=Content-Type $text/html
FilterChain SSI
```

Server side Includes (SSI)

The same as the above but dispatching on handler (classic SSI behaviour; `.shtml` files get processed).

```
FilterProvider SSI INCLUDES Handler server-parsed
FilterChain SSI
```

Emulating `mod_gzip` with `mod_deflate`

Insert INFLATE filter only if "gzip" is NOT in the Accept-Encoding header. This filter runs with `CONTENT_SET`.

```
FilterDeclare gzip CONTENT_SET
FilterProvider gzip inflate req=Accept-Encoding !$gzip
FilterChain gzip
```

Image Downsampling

Suppose we want to downsample all web images, and have filters for GIF, JPEG and PNG.

```
FilterProvider unpack jpeg_unpack Content-Type $image/jpeg
FilterProvider unpack gif_unpack Content-Type $image/gif
FilterProvider unpack png_unpack Content-Type $image/png

FilterProvider downsample downsample_filter Content-Type $image
FilterProtocol downsample "change=yes"

FilterProvider repack jpeg_pack Content-Type $image/jpeg
FilterProvider repack gif_pack Content-Type $image/gif
FilterProvider repack png_pack Content-Type $image/png
<Location /image-filter>
```

```
FilterChain unpack downsample repack  
</Location>
```



Filtering

Historically, each filter is responsible for ensuring that whatever changes it makes are correctly represented in the HTTP response headers, and that it does not run when it would make an illegal change. This imposes a burden on filter authors to re-implement some common functionality in every filter:

- Many filters will change the content, invalidating existing content tags, checksums, hashes, and lengths.
- Filters that require an entire, unbroken response in input need to ensure they don't get byteranges from a backend.
- Filters that transform output in a filter need to ensure they don't violate a `Cache-Control: no-transform` header from the backend.
- Filters may make responses uncacheable.

`mod_filter` aims to offer generic handling of these details of filter implementation, reducing the complexity required of content filter modules. This is work-in-progress; the `FilterProtocol` implements some of this functionality for back-compatibility with Apache 2.0 modules. For `httpd 2.1` and later, the `ap_register_output_filter_protocol` and `ap_filter_protocol` API enables filter modules to declare their own behaviour.

At the same time, `mod_filter` should not interfere with a filter that wants to handle all aspects of the protocol. By default (i.e. in the absence of any `FilterProtocol` directives), `mod_filter` will leave the headers untouched.

At the time of writing, this feature is largely untested, as modules in common use are designed to work with 2.0. Modules using it should test it carefully.



Description:	Configure the filter chain
Syntax:	FilterChain [+=-@!] <i>filter-name</i> ...
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Base
Module:	mod_filter

This configures an actual filter chain, from declared filters. **FilterChain** takes any number of arguments, each optionally preceded with a single-character control that determines what to do:

+*filter-name*

Add *filter-name* to the end of the filter chain

@*filter-name*

Insert *filter-name* at the start of the filter chain

-*filter-name*

Remove *filter-name* from the filter chain

=*filter-name*

Empty the filter chain and insert *filter-name*

!

Empty the filter chain

filter-name

Equivalent to +*filter-name*



Description:	Declare a smart filter
Syntax:	<code>FilterDeclare <i>filter-name</i> [<i>type</i>]</code>
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Base
Module:	mod_filter

This directive declares an output filter together with a header or environment variable that will determine runtime configuration. The first argument is a *filter-name* for use in [FilterProvider](#), [FilterChain](#) and [FilterProtocol](#) directives.

The final (optional) argument is the type of filter, and takes values of `ap_filter_type` - namely RESOURCE (the default), CONTENT_SET, PROTOCOL, TRANSCODE, CONNECTION or NETWORK.



Description:	Deal with correct HTTP protocol handling
Syntax:	FilterProtocol <i>filter-name</i> [<i>provider-name</i>] <i>proto-flags</i>
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Base
Module:	mod_filter

This directs [mod_filter](#) to deal with ensuring the filter doesn't run when it shouldn't, and that the HTTP response headers are correctly set taking into account the effects of the filter.

There are two forms of this directive. With three arguments, it applies specifically to a *filter-name* and a *provider-name* for that filter. With two arguments it applies to a *filter-name* whenever the filter runs *any* provider.

proto-flags is one or more of

change=yes

The filter changes the content, including possibly the content length

change=1:1

The filter changes the content, but will not change the content length

byteranges=no

The filter cannot work on byteranges and requires complete input

proxy=no

The filter should not run in a proxy context

proxy=transform

The filter transforms the response in a manner incompatible with the HTTP Cache-Control: no-transform header.

cache=no

The filter renders the output uncacheable (eg by introducing randomised content changes)



Description:	Register a content filter
Syntax:	FilterProvider <i>filter-name</i> <i>provider-name</i> [req resp env]= <i>dispatch</i> <i>match</i>
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Base
Module:	mod_filter

This directive registers a *provider* for the smart filter. The provider will be called if and only if the *match* declared here matches the value of the header or environment variable declared as *dispatch*.

provider-name must have been registered by loading a module that registers the name with `ap_register_output_filter`.

The *dispatch* argument is a string with optional `req=`, `resp=` or `env=` prefix causing it to dispatch on (respectively) the request header, response header, or environment variable named. In the absence of a prefix, it defaults to a response header. A special case is the word `handler`, which causes `mod_filter` to dispatch on the content handler.

The *match* argument specifies a match that will be applied to the filter's *dispatch* criterion. The *match* may be a string match (exact match or substring), a `regex`, an integer (greater, less than or equals), or unconditional. The first characters of the *match* argument determines this:

First, if the first character is an exclamation mark (!), this reverses the rule, so the provider will be used if and only if the match *fails*.

Second, it interprets the first character excluding any leading ! as follows:

Character	Description
<i>(none)</i>	exact match
\$	substring match
/	regex match (delimited by a second /)
=	integer equality
<	integer less-than
<=	integer less-than or equal
>	integer greater-than
>=	integer greater-than or equal
*	Unconditional match



Description:	Get debug/diagnostic information from mod_filter
Syntax:	FilterTrace <i>filter-name level</i>
Context:	server config, virtual host, directory
Status:	Base
Module:	mod_filter

This directive generates debug information from [mod_filter](#). It is designed to help test and debug providers (filter modules), although it may also help with [mod_filter](#) itself.

The debug output depends on the *level* set:

0 (default)

No debug information is generated.

1

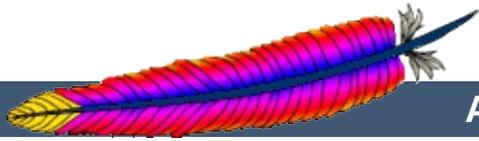
[mod_filter](#) will record buckets and brigades passing through the filter to the error log, before the provider has processed them. This is similar to the information generated by [mod_diagnostics](#).

2 (not yet implemented)

Will dump the full data passing through to a tempfile before the provider. **For single-user debug only**; this will not support concurrent hits.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_headers

Description:	Customization of HTTP request and response headers
Status:	Extension
Module Identifier:	headers_module
Source File:	mod_headers.c

Summary

This module provides directives to control and modify HTTP request and response headers. Headers can be merged, replaced or removed.



The directives provided by `mod_headers` can occur almost anywhere within the server configuration, and can be limited in scope by enclosing them in [configuration sections](#).

Order of processing is important and is affected both by the order in the configuration file and by placement in [configuration sections](#). These two directives have a different effect if reversed:

```
RequestHeader append MirrorID "mirror 12"  
RequestHeader unset MirrorID
```

This way round, the `MirrorID` header is not set. If reversed, the `MirrorID` header is set to "mirror 12".



`mod_headers` can be applied either early or late in the request. The normal mode is late, when *Request* Headers are set immediately before running the content generator and *Response* Headers just as the response is sent down the wire. Always use Late mode in an operational server.

Early mode is designed as a test/debugging aid for developers. Directives defined using the `early` keyword are set right at the beginning of processing the request. This means they can be used to simulate different requests and set up test cases, but it also means that headers may be changed at any time by other modules before generating a Response.

Because early directives are processed before the request path's configuration is traversed, early headers can only be set in a main server or virtual host context. Early directives cannot depend on a request path, so they will fail in contexts such as `<Directory>` or `<Location>`.



1. Copy all request headers that begin with "TS" to the response headers:

```
Header echo ^TS
```

2. Add a header, MyHeader, to the response including a timestamp for when the request was received and how long it took to begin serving the request. This header can be used by the client to intuit load on the server or in isolating bottlenecks between the client and the server.

```
Header set MyHeader "%D %t"
```

results in this header being added to the response:

```
MyHeader: D=3775428 t=991424704447256
```

3. Say hello to Joe

```
Header set MyHeader "Hello Joe. It took %D microseconds \\  
for Apache to serve this request."
```

results in this header being added to the response:

```
MyHeader: Hello Joe. It took D=3775428 microseconds for Apache to  
serve this request.
```

4. Conditionally send MyHeader on the response if and only if header MyRequestHeader is present on the request. This is useful for constructing headers in response to some client stimulus. Note that this example requires the services of the [mod_setenvif](#) module.

```
SetEnvIf MyRequestHeader myvalue HAVE_MyRequestHeader  
Header set MyHeader "%D %t mytext" env=HAVE_MyRequestHeader
```

If the header `MyRequestHeader: myvalue` is present on the HTTP request, the response will contain the following header:

```
MyHeader: D=3775428 t=991424704447256 mytext
```

5. Enable DAV to work with Apache running HTTP through SSL hardware ([problem description](#)) by replacing `https:` with `http:` in the *Destination* header:

```
RequestHeader edit Destination ^https: http: early
```

6. Set the same header value under multiple nonexclusive conditions, but do not duplicate the value in the final header. If all of the following conditions applied to a request (i.e., if the `CGI`, `NO_CACHE` and `NO_STORE` environment variables all existed for the request):

```
Header merge Cache-Control no-cache env=CGI
Header merge Cache-Control no-cache env=NO_CACHE
Header merge Cache-Control no-store env=NO_STORE
```

then the response would contain the following header:

```
Cache-Control: no-cache, no-store
```

If `append` was used instead of `merge`, then the response would contain the following header:

```
Cache-Control: no-cache, no-cache, no-store
```



Description:	Configure HTTP response headers
Syntax:	Header [<i>condition</i>] set append merge add unset echo edit header [<i>value</i>] [<i>replacement</i>] [early env= [!] <i>variable</i>]
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_headers
Compatibility:	The merge argument is available in version 2.2.9 and later. The edit argument is available in version 2.2.4 and later.

This directive can replace, merge or remove HTTP response headers. The header is modified just after the content handler and output filters are run, allowing outgoing headers to be modified.

The optional *condition* argument determines which internal table of responses headers this directive will operate against. Other components of the server may have stored their response headers in either the table that corresponds to `onsuccess` or the table that corresponds to `always`. "Always" in this context refers to whether headers you add will be sent during both a successful and unsuccessful response, but if your action is a function of an existing header, you will have to read on for further complications.

The default value of `onsuccess` may need to be changed to `always` under the circumstances similar to those listed below. Note also that repeating this directive with both conditions makes sense in some scenarios because `always` is not a superset of `onsuccess` with respect to existing headers:

- You're adding a header to a non-success (non-2xx) response, such

as a redirect, in which case only the table corresponding to `always` is used in the ultimate response.

- You're modifying or removing a header generated by a CGI script, in which case the CGI scripts are in the table corresponding to `always` and not in the default table.
- You're modifying or removing a header generated by some piece of the server but that header is not being found by the default `onsuccess` condition.

The action it performs is determined by the first argument (second argument if a *condition* is specified). This can be one of the following values:

set

The response header is set, replacing any previous header with this name. The *value* may be a format string.

append

The response header is appended to any existing header of the same name. When a new value is merged onto an existing header it is separated from the existing header with a comma. This is the HTTP standard way of giving a header multiple values.

merge

The response header is appended to any existing header of the same name, unless the value to be appended already appears in the header's comma-delimited list of values. When a new value is merged onto an existing header it is separated from the existing header with a comma. This is the HTTP standard way of giving a header multiple values. Values are compared in a case sensitive manner, and after all format specifiers have been processed. Values in double quotes are considered different from otherwise identical unquoted values. *Available in version 2.2.9 and later.*

add

The response header is added to the existing set of headers, even if

this header already exists. This can result in two (or more) headers having the same name. This can lead to unforeseen consequences, and in general `set`, `append` or `merge` should be used instead.

unset

The response header of this name is removed, if it exists. If there are multiple headers of the same name, all will be removed. *value* must be omitted.

echo

Request headers with this name are echoed back in the response headers. *header* may be a [regular expression](#). *value* must be omitted.

edit

If this response header exists, its value is transformed according to a [regular expression](#) search-and-replace. The *value* argument is a [regular expression](#), and the *replacement* is a replacement string, which may contain backreferences. *Available in version 2.2.4 and later.*

This argument is followed by a *header* name, which can include the final colon, but it is not required. Case is ignored for `set`, `append`, `merge`, `add`, `unset` and `edit`. The *header* name for `echo` is case sensitive and may be a [regular expression](#).

For `set`, `append`, `merge` and `add` a *value* is specified as the next argument. If *value* contains spaces, it should be surrounded by double quotes. *value* may be a character string, a string containing format specifiers or a combination of both. The following format specifiers are supported in *value*:

Format	Description
%%	The percent sign
%t	The time the request was received in Universal Coordinated Time since the epoch (Jan. 1, 1970)

	measured in microseconds. The value is preceded by t=.
%D	The time from when the request was received to the time the headers are sent on the wire. This is a measure of the duration of the request. The value is preceded by D=. The value is measured in microseconds.
% {FOOBAR}e	The contents of the environment variable FOOBAR.
% {FOOBAR}s	The contents of the SSL environment variable FOOBAR, if <code>mod_ssl</code> is enabled.

Note

The %s format specifier is only available in Apache 2.1 and later; it can be used instead of %e to avoid the overhead of enabling `SSLOptions +StdEnvVars`. If `SSLOptions +StdEnvVars` must be enabled anyway for some other reason, %e will be more efficient than %s.

For edit there is both a *value* argument which is a [regular expression](#), and an additional *replacement* string.

The **Header** directive may be followed by an an additional argument, which may be used to specify conditions under which the action will be taken, or may be the keyword `early` to specify [early processing](#). If the [environment variable](#) specified in the `env=...` argument exists (or if the environment variable does not exist and `env=!...` is specified) then the action specified by the **Header** directive will take effect. Otherwise, the directive will have no effect on the request.

Except in [early](#) mode, the **Header** directives are processed just before the response is sent to the network. This means that it is possible to set and/or override most headers, except for some headers added by the HTTP header filter. Prior to 2.2.12, it was not possible to change the Content-Type header with this directive.



RequestHeader Directive

Description:	Configure HTTP request headers
Syntax:	RequestHeader set append merge add unset edit <i>header</i> [<i>value</i>] [<i>replacement</i>] [early env= [!] <i>variable</i>]
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_headers
Compatibility:	The merge argument is available in version 2.2.9 and later. The edit argument is available in version 2.2.4 and later.

This directive can replace, merge, change or remove HTTP request headers. The header is modified just before the content handler is run, allowing incoming headers to be modified. The action it performs is determined by the first argument. This can be one of the following values:

set

The request header is set, replacing any previous header with this name

append

The request header is appended to any existing header of the same name. When a new value is merged onto an existing header it is separated from the existing header with a comma. This is the HTTP standard way of giving a header multiple values.

merge

The request header is appended to any existing header of the same name, unless the value to be appended already appears in the existing header's comma-delimited list of values. When a new value is merged onto an existing header it is separated from the existing header with a comma. This is the HTTP standard way of giving a

header multiple values. Values are compared in a case sensitive manner, and after all format specifiers have been processed. Values in double quotes are considered different from otherwise identical unquoted values. *Available in version 2.2.9 and later.*

add

The request header is added to the existing set of headers, even if this header already exists. This can result in two (or more) headers having the same name. This can lead to unforeseen consequences, and in general set, append or merge should be used instead.

unset

The request header of this name is removed, if it exists. If there are multiple headers of the same name, all will be removed. *value* must be omitted.

edit

If this request header exists, its value is transformed according to a [regular expression](#) search-and-replace. The *value* argument is a [regular expression](#), and the *replacement* is a replacement string, which may contain backreferences. *Available in version 2.2.4 and later.*

This argument is followed by a header name, which can include the final colon, but it is not required. Case is ignored. For set, append, merge and add a *value* is given as the third argument. If a *value* contains spaces, it should be surrounded by double quotes. For unset, no *value* should be given. *value* may be a character string, a string containing format specifiers or a combination of both. The supported format specifiers are the same as for the [Header](#), please have a look there for details. For edit both a *value* and a *replacement* are required, and are a [regular expression](#) and a replacement string respectively.

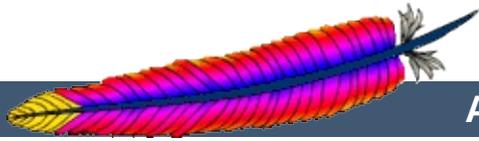
The [RequestHeader](#) directive may be followed by an additional argument, which may be used to specify conditions under which the action will be taken, or may be the keyword `early` to specify [early](#)

[processing](#). If the [environment variable](#) specified in the `env= . . .` argument exists (or if the environment variable does not exist and `env=! . . .` is specified) then the action specified by the [RequestHeader](#) directive will take effect. Otherwise, the directive will have no effect on the request.

Except in [early](#) mode, the [RequestHeader](#) directive is processed just before the request is run by its handler in the fixup phase. This should allow headers generated by the browser, or by Apache input filters to be overridden or modified.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_ident

Description:	RFC 1413 ident lookups
Status:	Extension
Module Identifier:	ident_module
Source File:	mod_ident.c
Compatibility:	Available in Apache 2.1 and later

Summary

This module queries an [RFC 1413](#) compatible daemon on a remote host to look up the owner of a connection.

See also

[mod_log_config](#)



IdentityCheck Directive

Description:	Enables logging of the RFC 1413 identity of the remote user
Syntax:	IdentityCheck On Off
Default:	IdentityCheck Off
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_ident
Compatibility:	Moved out of core in Apache 2.1

This directive enables [RFC 1413](#)-compliant logging of the remote user name for each connection, where the client machine runs `identd` or something similar. This information is logged in the access log using the `%. . . 1` [format string](#).

The information should not be trusted in any way except for rudimentary usage tracking.

Note that this can cause serious latency problems accessing your server since every request requires one of these lookups to be performed. When firewalls or proxy servers are involved, each lookup might possibly fail and add a latency duration as defined by the [IdentityCheckTimeout](#) directive to each hit. So in general this is not very useful on public servers accessible from the Internet.

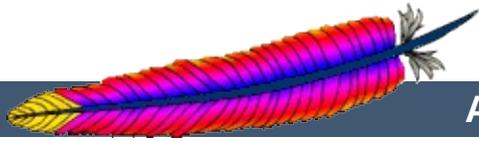


Description:	Determines the timeout duration for ident requests
Syntax:	<code>IdentityCheckTimeout</code> <i>seconds</i>
Default:	<code>IdentityCheckTimeout</code> 30
Context:	server config, virtual host, directory
Status:	Extension
Module:	<code>mod_ident</code>

This directive specifies the timeout duration of an ident request. The default value of 30 seconds is recommended by [RFC 1413](#), mainly because of possible network latency. However, you may want to adjust the timeout value according to your local network speed.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_imagemap

Description:	Server-side imagemap processing
Status:	Base
Module Identifier:	imagemap_module
Source File:	mod_imagemap.c

Summary

This module processes `.map` files, thereby replacing the functionality of the `imagemap` CGI program. Any directory or document type configured to use the handler `imap-file` (using either [AddHandler](#) or [SetHandler](#)) will be processed by this module.

The following directive will activate files ending with `.map` as imagemap files:

```
AddHandler imap-file map
```

Note that the following is still supported:

```
AddType application/x-httpd-imap map
```

However, we are trying to phase out "magic MIME types" so we are deprecating this method.



The imagemap module adds some new features that were not possible with previously distributed imagemap programs.

- URL references relative to the Referer: information.
- Default <base> assignment through a new map directive base.
- No need for `imagemap.conf` file.
- Point references.
- Configurable generation of imagemap menus.



The lines in the imagemap files can have one of several formats:

```
directive value [x,y ...]
directive value "Menu text" [x,y ...]
directive value x,y ... "Menu text"
```

The directive is one of `base`, `default`, `poly`, `circle`, `rect`, or `point`. The value is an absolute or relative URL, or one of the special values listed below. The coordinates are `x,y` pairs separated by whitespace. The quoted text is used as the text of the link if a imagemap menu is generated. Lines beginning with '#' are comments.

Imagemap File Directives

There are six directives allowed in the imagemap file. The directives can come in any order, but are processed in the order they are found in the imagemap file.

base Directive

Has the effect of `<base href="value">` . The non-absolute URLs of the map-file are taken relative to this value. The base directive overrides [ImapBase](#) as set in a `.htaccess` file or in the server configuration files. In the absence of an [ImapBase](#) configuration directive, base defaults to `http://server_name/`.

`base_uri` is synonymous with `base`. Note that a trailing slash on the URL is significant.

default Directive

The action taken if the coordinates given do not fit any of the `poly`, `circle` or `rect` directives, and there are no `point` directives. Defaults to `nocontent` in the absence of an [ImapDefault](#) configuration setting, causing a status code of 204 No Content to be returned. The client should keep the same page displayed.

poly Directive

Takes three to one-hundred points, and is obeyed if the user selected coordinates fall within the polygon defined by these points.

circle

Takes the center coordinates of a circle and a point on the circle. Is obeyed if the user selected point is with the circle.

rect Directive

Takes the coordinates of two opposing corners of a rectangle. Obeyed if the point selected is within this rectangle.

point Directive

Takes a single point. The point directive closest to the user selected point is obeyed if no other directives are satisfied. Note that default will not be followed if a point directive is present and valid coordinates are given.

Values

The values for each of the directives can be any of the following:

a URL

The URL can be relative or absolute URL. Relative URLs can contain '..' syntax and will be resolved relative to the base value.

base itself will not be resolved according to the current value. A statement base `mailto:` will work properly, though.

map

Equivalent to the URL of the imagemap file itself. No coordinates are sent with this, so a menu will be generated unless [ImapMenu](#) is set to none.

menu

Synonymous with map.

referer

Equivalent to the URL of the referring document. Defaults to `http://servername/` if no `Referer` header was present.

nocontent

Sends a status code of 204 No Content, telling the client to keep the same page displayed. Valid for all but base.

error

Fails with a 500 Server Error. Valid for all but base, but sort of silly for anything but default.

Coordinates

0,0 200,200

A coordinate consists of an x and a y value separated by a comma. The coordinates are separated from each other by whitespace. To accommodate the way Lynx handles imagemaps, should a user select the coordinate 0, 0, it is as if no coordinate had been selected.

Quoted Text

"Menu Text"

After the value or after the coordinates, the line optionally may contain text within double quotes. This string is used as the text for the link if a menu is generated:

```
<a href="http://example.com/">Menu text</a>
```

If no quoted text is present, the name of the link will be used as the text:

```
<a href="http://example.com/">http://example.com</a>
```

If you want to use double quotes within this text, you have to write them as `"` ; .



```
#Comments are printed in a 'formatted' or 'semiformatted' menu.  
#And can contain html tags. <hr>  
base referer  
poly map "Could I have a menu, please?" 0,0 0,10 10,10 10,0  
rect .. 0,0 77,27 "the directory of the referer"  
circle http://www.inetnebr.example.com/lincoln/feedback/ 195,0 305,27  
rect another_file "in same directory as referer" 306,0 419,27  
point http://www.zyzyva.example.com/ 100,100  
point http://www.tripod.example.com/ 200,200  
rect mailto:nate@tripod.example.com 100,150 200,0 "Bugs?"
```



HTML example

```
<a href="/maps/imagemap1.map">  
    
</a>
```

XHTML example

```
<a href="/maps/imagemap1.map">  
    
</a>
```



Description:	Default base for imagemap files
Syntax:	ImapBase map referer URL
Default:	ImapBase http://servername/
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_imagemap

The **ImapBase** directive sets the default base used in the imagemap files. Its value is overridden by a base directive within the imagemap file. If not present, the base defaults to `http://servername/`.

See also

- [UseCanonicalName](#)



ImapDefault Directive

Description:	Default action when an imagemap is called with coordinates that are not explicitly mapped
Syntax:	ImapDefault error nocontent map referer <i>URL</i>
Default:	ImapDefault nocontent
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_imagemap

The `ImapDefault` directive sets the default default used in the imagemap files. Its value is overridden by a default directive within the imagemap file. If not present, the default action is nocontent, which means that a 204 No Content is sent to the client. In this case, the client should continue to display the original page.



Description:	Action if no coordinates are given when calling an imagemap
Syntax:	ImapMenu none formatted semiformatted unformatted
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_imagemap

The **ImapMenu** directive determines the action taken if an imagemap file is called without valid coordinates.

none

If ImapMenu is none, no menu is generated, and the default action is performed.

formatted

A formatted menu is the simplest menu. Comments in the imagemap file are ignored. A level one header is printed, then an hrule, then the links each on a separate line. The menu has a consistent, plain look close to that of a directory listing.

semiformatted

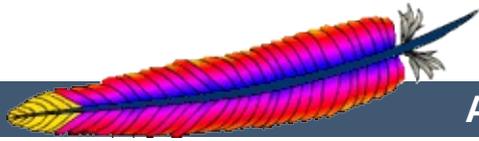
In the semiformatted menu, comments are printed where they occur in the imagemap file. Blank lines are turned into HTML breaks. No header or hrule is printed, but otherwise the menu is the same as a formatted menu.

unformatted

Comments are printed, blank lines are ignored. Nothing is printed that does not appear in the imagemap file. All breaks and headers must be included as comments in the imagemap file. This gives you the most flexibility over the appearance of your menus, but requires you to treat your map files as HTML instead of plaintext.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_include

Description:	Server-parsed html documents (Server Side Includes)
Status:	Base
Module Identifier:	include_module
Source File:	mod_include.c
Compatibility:	Implemented as an output filter since Apache 2.0

Summary

This module provides a filter which will process files before they are sent to the client. The processing is controlled by specially formatted SGML comments, referred to as *elements*. These elements allow conditional text, the inclusion of other files or programs, as well as the setting and printing of environment variables.

See also

[Options](#)

[AcceptPathInfo](#)

[Filters](#)

[SSI Tutorial](#)



Enabling Server Side Includes

Server Side Includes are implemented by the `INCLUDES` [filter](#). If documents containing server-side include directives are given the extension `.shtml`, the following directives will make Apache parse them and assign the resulting document the mime type of `text/html`:

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

The following directive must be given for the directories containing the `shtml` files (typically in a `<Directory>` section, but this directive is also valid in `.htaccess` files if `AllowOverride Options` is set):

```
Options +Includes
```

For backwards compatibility, the server - parsed [handler](#) also activates the `INCLUDES` filter. As well, Apache will activate the `INCLUDES` filter for any document with mime type `text/x-server-parsed-html` or `text/x-server-parsed-html3` (and the resulting output will have the mime type `text/html`).

For more information, see our [Tutorial on Server Side Includes](#).



Files processed for server-side includes no longer accept requests with PATH_INFO (trailing pathname information) by default. You can use the [AcceptPathInfo](#) directive to configure the server to accept requests with PATH_INFO.



The document is parsed as an HTML document, with special commands embedded as SGML comments. A command has the syntax:

```
<!--#element attribute=value attribute=value ... -->
```

The value will often be enclosed in double quotes, but single quotes (') and backticks (`) are also possible. Many commands only allow a single attribute-value pair. Note that the comment terminator (- ->) should be preceded by whitespace to ensure that it isn't considered part of an SSI token. Note that the leading < ! - - # is *one* token and may not contain any whitespaces.

The allowed elements are listed in the following table:

Element	Description
config	configure output formats
echo	print variables
exec	execute external programs
fsize	print size of a file
flastmod	print last modification time of a file
include	include a file
printenv	print all available variables
set	set a value of a variable

SSI elements may be defined by modules other than [mod_include](#). In fact, the [exec](#) element is provided by [mod_cgi](#), and will only be available if this module is loaded.

The config Element

This command controls various aspects of the parsing. The valid attributes are:

echormsg (Apache 2.1 and later)

The value is a message that is sent back to the client if the [echo](#) element attempts to echo an undefined variable. This overrides any [SSIUndefinedEcho](#) directives.

errmsg

The value is a message that is sent back to the client if an error occurs while parsing the document. This overrides any [SSIErrorMsg](#) directives.

sizefmt

The value sets the format to be used when displaying the size of a file. Valid values are `bytes` for a count in bytes, or `abbrev` for a count in Kb or Mb as appropriate, for example a size of 1024 bytes will be printed as "1K".

timefmt

The value is a string to be used by the `strftime(3)` library routine when printing dates.

The echo Element

This command prints one of the [include variables](#) defined below. If the variable is unset, the result is determined by the [SSIUndefinedEcho](#) directive. Any dates printed are subject to the currently configured `timefmt`.

Attributes:

var

The value is the name of the variable to print.

encoding

Specifies how Apache should encode special characters contained in the variable before outputting them. If set to `none`, no encoding will be done. If set to `url`, then URL encoding (also known as %-encoding; this is appropriate for use within URLs in links, etc.) will be

performed. At the start of an echo element, the default is set to `entity`, resulting in entity encoding (which is appropriate in the context of a block-level HTML element, e.g. a paragraph of text). This can be changed by adding an `encoding` attribute, which will remain in effect until the next `encoding` attribute is encountered or the element ends, whichever comes first.

The `encoding` attribute must *precede* the corresponding `var` attribute to be effective, and only special characters as defined in the ISO-8859-1 character encoding will be encoded. This encoding process may not have the desired result if a different character encoding is in use.

In order to avoid cross-site scripting issues, you should *always* encode user supplied data.

The `exec` Element

The `exec` command executes a given shell command or CGI script. It requires `mod_cgi` to be present in the server. If `Options IncludesNOEXEC` is set, this command is completely disabled. The valid attributes are:

`cgi`

The value specifies a (%-encoded) URL-path to the CGI script. If the path does not begin with a slash (/), then it is taken to be relative to the current document. The document referenced by this path is invoked as a CGI script, even if the server would not normally recognize it as such. However, the directory containing the script must be enabled for CGI scripts (with `ScriptAlias` or `Options ExecCGI`).

The CGI script is given the `PATH_INFO` and query string (`QUERY_STRING`) of the original request from the client; these

cannot be specified in the URL path. The include variables will be available to the script in addition to the standard [CGI](#) environment.

Example

```
<!--#exec cgi="/cgi-bin/example.cgi" -->
```

If the script returns a `Location:` header instead of output, then this will be translated into an HTML anchor.

The [include virtual](#) element should be used in preference to `exec cgi`. In particular, if you need to pass additional arguments to a CGI program, using the query string, this cannot be done with `exec cgi`, but can be done with `include virtual`, as shown here:

```
<!--#include virtual="/cgi-bin/example.cgi?argument=value" -->
```

cmd

The server will execute the given string using `/bin/sh`. The [include variables](#) are available to the command, in addition to the usual set of CGI variables.

The use of [#include virtual](#) is almost always preferred to using either `#exec cgi` or `#exec cmd`. The former (`#include virtual`) uses the standard Apache sub-request mechanism to include files or scripts. It is much better tested and maintained.

In addition, on some platforms, like Win32, and on unix when using [suexec](#), you cannot pass arguments to a command in an `exec` directive, or otherwise include spaces in the command. Thus, while the following will work under a non-`suexec` configuration on unix, it will not produce the desired result under Win32, or when running `suexec`:

```
<!--#exec cmd="perl /path/to/perlscript arg1 arg2" -->
```

The `fsize` Element

This command prints the size of the specified file, subject to the `sizefmt` format specification. Attributes:

file

The value is a path relative to the directory containing the current document being parsed.

virtual

The value is a (%-encoded) URL-path. If it does not begin with a slash (/) then it is taken to be relative to the current document. Note, that this does *not* print the size of any CGI output, but the size of the CGI script itself.

The `flastmod` Element

This command prints the last modification date of the specified file, subject to the `timefmt` format specification. The attributes are the same as for the [fsize](#) command.

The `include` Element

This command inserts the text of another document or file into the parsed file. Any included file is subject to the usual access control. If the directory containing the parsed file has [Options](#) `IncludesNOEXEC` set, then only documents with a text [MIME-type](#) (text/plain, text/html etc.) will be included. Otherwise CGI scripts are invoked as normal using the complete URL given in the command, including any query string.

An attribute defines the location of the document; the inclusion is done for each attribute given to the include command. The valid attributes are:

file

The value is a path relative to the directory containing the current document being parsed. It cannot contain `../`, nor can it be an absolute path. Therefore, you cannot include files that are outside of the document root, or above the current document in the directory structure. The `virtual` attribute should always be used in preference to this one.

virtual

The value is a (%-encoded) URL-path. The URL cannot contain a scheme or hostname, only a path and an optional query string. If it does not begin with a slash (/) then it is taken to be relative to the current document.

A URL is constructed from the attribute, and the output the server would return if the URL were accessed by the client is included in the parsed output. Thus included files can be nested.

If the specified URL is a CGI program, the program will be executed and its output inserted in place of the directive in the parsed file. You may include a query string in a CGI url:

```
<!--#include virtual="/cgi-bin/example.cgi?argument=value" -->
```

`include virtual` should be used in preference to `exec cgi` to include the output of CGI programs into an HTML document.

The printenv Element

This prints out a listing of all existing variables and their values. Special characters are entity encoded (see the [echo](#) element for details) before being output. There are no attributes.

Example

```
<!--#printenv -->
```

The set Element

This sets the value of a variable. Attributes:

var

The name of the variable to set.

value

The value to give a variable.

Example

```
<!--#set var="category" value="help" -->
```



In addition to the variables in the standard CGI environment, these are available for the `echo` command, for `if` and `elif`, and to any program invoked by the document.

DATE_GMT

The current date in Greenwich Mean Time.

DATE_LOCAL

The current date in the local time zone.

DOCUMENT_NAME

The filename (excluding directories) of the document requested by the user.

DOCUMENT_URI

The (%-decoded) URL path of the document requested by the user. Note that in the case of nested include files, this is *not* the URL for the current document. Note also that if the URL is modified internally (e.g. by an [alias](#) or [directoryindex](#)), the modified URL is shown.

LAST_MODIFIED

The last modification date of the document requested by the user.

QUERY_STRING_UNESCAPED

If a query string is present, this variable contains the (%-decoded) query string, which is *escaped* for shell usage (special characters like `&` etc. are preceded by backslashes).



Variable Substitution

Variable substitution is done within quoted strings in most cases where they may reasonably occur as an argument to an SSI directive. This includes the `config`, `exec`, `flastmod`, `fsize`, `include`, `echo`, and `set` directives, as well as the arguments to conditional operators. You can insert a literal dollar sign into the string using backslash quoting:

```
<!--#if expr="$a = \$test" -->
```

If a variable reference needs to be substituted in the middle of a character sequence that might otherwise be considered a valid identifier in its own right, it can be disambiguated by enclosing the reference in braces, *a la* shell substitution:

```
<!--#set var="Zed" value="${REMOTE_HOST}_${REQUEST_METHOD}" -->
```

This will result in the Zed variable being set to "X_Y" if REMOTE_HOST is "X" and REQUEST_METHOD is "Y".

The below example will print "in foo" if the DOCUMENT_URI is /foo/file.html, "in bar" if it is /bar/file.html and "in neither" otherwise:

```
<!--#if expr='"$DOCUMENT_URI" = "/foo/file.html"' -->
  in foo
<!--#elif expr='"$DOCUMENT_URI" = "/bar/file.html"' -->
  in bar
<!--#else -->
  in neither
<!--#endif -->
```



The basic flow control elements are:

```
<!--#if expr="test_condition" -->  
<!--#elif expr="test_condition" -->  
<!--#else -->  
<!--#endif -->
```

The `if` element works like an if statement in a programming language. The test condition is evaluated and if the result is true, then the text until the next `elif`, `else` or `endif` element is included in the output stream.

The `elif` or `else` statements are used to put text into the output stream if the original `test_condition` was false. These elements are optional.

The `endif` element ends the `if` element and is required.

`test_condition` is one of the following:

string

true if *string* is not empty

-A string

true if the URL represented by the string is accessible by configuration, false otherwise. This test only has an effect if `SSIEnableAccess` is on. This is useful where content on a page is to be hidden from users who are not authorized to view the URL, such as a link to that URL. Note that the URL is only tested for whether access would be granted, not whether the URL exists.

Example

```
<!--#if expr="-A /private" -->  
  Click <a href="/private">here</a> to access private  
  information.  
<!--#endif -->
```

string1 = string2

string1* == *string2
string1* != *string2

Compare *string1* with *string2*. If *string2* has the form */string2/* then it is treated as a regular expression. Regular expressions are implemented by the [PCRE](#) engine and have the same syntax as those in [perl 5](#). Note that == is just an alias for = and behaves exactly the same way.

If you are matching positive (= or ==), you can capture grouped parts of the regular expression. The captured parts are stored in the special variables \$1 .. \$9.

Example

```
<!--#if expr="$QUERY_STRING = /^sid=([a-zA-Z0-9]+)/" -->  
  <!--#set var="session" value="$1" -->  
<!--#endif -->
```

string1* < *string2
string1* <= *string2
string1* > *string2
string1* >= *string2

Compare *string1* with *string2*. Note, that strings are compared *literally* (using `strcmp(3)`). Therefore the string "100" is less than "20".

(*test_condition*)
true if *test_condition* is true

! *test_condition*
true if *test_condition* is false

test_condition1* && *test_condition2
true if both *test_condition1* and *test_condition2* are true

test_condition1* || *test_condition2
true if either *test_condition1* or *test_condition2* is true

"=" and "!=" bind more tightly than "&&" and "| |". "!" binds most tightly. Thus, the following are equivalent:

```
<!--#if expr="$a = test1 && $b = test2" -->  
<!--#if expr="($a = test1) && ($b = test2)" -->
```

The boolean operators && and | | share the same priority. So if you want to bind such an operator more tightly, you should use parentheses.

Anything that's not recognized as a variable or an operator is treated as a string. Strings can also be quoted: 'string'. Unquoted strings can't contain whitespace (blanks and tabs) because it is used to separate tokens such as variables. If multiple strings are found in a row, they are concatenated using blanks. So,

```
string1 string2 results in string1 string2
```

and

```
'string1 string2' results in string1 string2.
```

Optimization of Boolean Expressions

If the expressions become more complex and slow down processing significantly, you can try to optimize them according to the evaluation rules:

- Expressions are evaluated from left to right
- Binary boolean operators (&& and | |) are short circuited wherever possible. In conclusion with the rule above that means, mod_include evaluates at first the left expression. If the left result is sufficient to determine the end result, processing stops here. Otherwise it evaluates the right side and computes the end result from both left and right results.
- Short circuit evaluation is turned off as long as there are regular

expressions to deal with. These must be evaluated to fill in the backreference variables (\$1 .. \$9).

If you want to look how a particular expression is handled, you can recompile `mod_include` using the `-DDEBUG_INCLUDE` compiler option. This inserts for every parsed expression tokenizer information, the parse tree and how it is evaluated into the output sent to the client.

Escaping slashes in regex strings

All slashes which are not intended to act as delimiters in your regex must be escaped. This is regardless of their meaning to the regex engine.



Description:	Enable the -A flag during conditional flow control processing.
Syntax:	<code>SSIEnableAccess on off</code>
Default:	<code>SSIEnableAccess off</code>
Context:	directory, .htaccess
Status:	Base
Module:	mod_include

The `SSIEnableAccess` directive controls whether the -A test is enabled during conditional flow control processing. `SSIEnableAccess` can take on the following values:

off

`<!--#if expr="-A /foo"-->` will be interpreted as a series of string and regular expression tokens, the -A has no special meaning.

on

`<!--#if expr="-A /foo"-->` will evaluate to false if the URL /foo is inaccessible by configuration, or true otherwise.



Description:	String that ends an include element
Syntax:	SSIEndTag <i>tag</i>
Default:	SSIEndTag " -->"
Context:	server config, virtual host
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

This directive changes the string that [mod_include](#) looks for to mark the end of an include element.

Example

```
SSIEndTag "%>"
```

See also

- [SSIStartTag](#)



Description:	Error message displayed when there is an SSI error
Syntax:	SSIErrorMsg <i>message</i>
Default:	SSIErrorMsg "[an error occurred while processing this directive]"
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

The `SSIErrorMsg` directive changes the error message displayed when `mod_include` encounters an error. For production servers you may consider changing the default error message to "`<!-- Error -->`" so that the message is not presented to the user.

This directive has the same effect as the `<!--#config errmsg=message -->` element.

Example

```
SSIErrorMsg "<!-- Error -->"
```



Description:	Controls whether ETags are generated by the server.
Syntax:	SSIETag on off
Default:	SSIETag off
Context:	directory, .htaccess
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.2.15 and later.

Under normal circumstances, a file filtered by `mod_include` may contain elements that are either dynamically generated, or that may have changed independently of the original file. As a result, by default the server is asked not to generate an ETag header for the response by adding `no-etag` to the request notes.

The `SSIETag` directive suppresses this behaviour, and allows the server to generate an ETag header. This can be used to enable caching of the output. Note that a backend server or dynamic content generator may generate an ETag of its own, ignoring `no-etag`, and this ETag will be passed by `mod_include` regardless of the value of this setting.

`SSIETag` can take on the following values:

off

`no-etag` will be added to the request notes, and the server is asked not to generate an ETag. Where a server ignores the value of `no-etag` and generates an ETag anyway, the ETag will be respected.

on

Existing ETags will be respected, and ETags generated by the server will be passed on in the response.



Description:	Controls whether Last -Modified headers are generated by the server.
Syntax:	SSILastModified on off
Default:	SSILastModified off
Context:	directory, .htaccess
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.2.15 and later.

Under normal circumstances, a file filtered by `mod_include` may contain elements that are either dynamically generated, or that may have changed independently of the original file. As a result, by default the Last -Modified header is stripped from the response.

The `SSILastModified` directive overrides this behaviour, and allows the Last -Modified header to be respected if already present, or set if the header is not already present. This can be used to enable caching of the output. `SSILastModified` can take on the following values:

off

The Last -Modified header will be stripped from responses, unless the `XBitHack` directive is set to `full` as described below.

on

The Last -Modified header will be respected if already present in a response, and added to the response if the response is a file and the header is missing. The `SSILastModified` directive takes precedence over `XBitHack`.



Description:	String that starts an include element
Syntax:	SSIStartTag <i>tag</i>
Default:	SSIStartTag "<! - -#"
Context:	server config, virtual host
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

This directive changes the string that [mod_include](#) looks for to mark an include element to process.

You may want to use this option if you have 2 servers parsing the output of a file each processing different commands (possibly at different times).

Example

```
SSIStartTag "<%"  
SSIEndTag "%>"
```

The example given above, which also specifies a matching [SSIEndTag](#), will allow you to use SSI directives as shown in the example below:

SSI directives with alternate start and end tags

```
<%printenv %>
```

See also

- [SSIEndTag](#)



Description:	Configures the format in which date strings are displayed
Syntax:	SSITimeFormat <i>formatstring</i>
Default:	SSITimeFormat "%A, %d-%b-%Y %H:%M:%S %Z"
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

This directive changes the format in which date strings are displayed when echoing DATE environment variables. The *formatstring* is as in `strftime(3)` from the C standard library.

This directive has the same effect as the `<!--#config timefmt=formatstring -->` element.

Example

```
SSITimeFormat "%R, %B %d, %Y"
```

The above directive would cause times to be displayed in the format "22:26, June 14, 2002".



Description:	String displayed when an unset variable is echoed
Syntax:	SSIUndefinedEcho <i>string</i>
Default:	SSIUndefinedEcho "(none)"
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.34 and later.

This directive changes the string that [mod_include](#) displays when a variable is not set and "echoed".

Example

```
SSIUndefinedEcho "<!-- undef -->"
```



Description:	Parse SSI directives in files with the execute bit set
Syntax:	XBitHack on off full
Default:	XBitHack off
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Base
Module:	mod_include

The **XBitHack** directive controls the parsing of ordinary html documents. This directive only affects files associated with the [MIME-type](#) text/html. **XBitHack** can take on the following values:

off

No special treatment of executable files.

on

Any text/html file that has the user-execute bit set will be treated as a server-parsed html document.

full

As for on but also test the group-execute bit. If it is set, then set the Last-modified date of the returned file to be the last modified time of the file. If it is not set, then no last-modified date is sent. Setting this bit allows clients and proxies to cache the result of the request.

Note

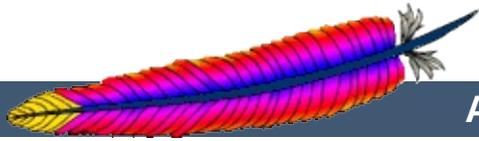
You would not want to use the full option, unless you assure the group-execute bit is unset for every SSI script which might `#include` a CGI or otherwise produces different output on each hit (or could potentially change on subsequent requests).

The [SSILastModified](#) directive takes precedence over the

[XBitHack](#) directive when [SSILastModified](#) is set to on.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_info

Description:	Provides a comprehensive overview of the server configuration
Status:	Extension
Module Identifier:	info_module
Source File:	mod_info.c

Summary

To configure `mod_info`, add the following to your `httpd.conf` file.

```
<Location /server-info>
  SetHandler server-info
</Location>
```

You may wish to use `mod_authz_host` inside the `<Location>` directive to limit access to your server configuration information:

```
<Location /server-info>
  SetHandler server-info
  Order deny,allow
  Deny from all
  Allow from yourcompany.com
</Location>
```

Once configured, the server information is obtained by accessing `http://your.host.example.com/server-info`



Once `mod_info` is loaded into the server, its handler capability is available in *all* configuration files, including per-directory files (e.g., `.htaccess`). This may have security-related ramifications for your site.

In particular, this module can leak sensitive information from the configuration directives of other Apache modules such as system paths, usernames/passwords, database names, etc. Therefore, this module should **only** be used in a controlled environment and always with caution.

You will probably want to use `mod_authz_host` to limit access to your server configuration information.

Access control

```
<Location /server-info>
  SetHandler server-info
  Order allow,deny
  # Allow access from server itself
  Allow from 127.0.0.1
  # Additionally, allow access from local workstation
  Allow from 192.168.1.17
</Location>
```



By default, the server information includes a list of all enabled modules, and for each module, a description of the directives understood by that module, the hooks implemented by that module, and the relevant directives from the current configuration.

Other views of the configuration information are available by appending a query to the `server-info` request. For example, `http://your.host.example.com/server-info?config` will show all configuration directives.

?<module-name>

Only information relevant to the named module

?config

Just the configuration directives, not sorted by module

?hooks

Only the list of Hooks each module is attached to

?list

Only a simple list of enabled modules

?server

Only the basic server information



KNOWN LIMITATIONS

`mod_info` provides its information by reading the parsed configuration, rather than reading the original configuration file. There are a few limitations as a result of the way the parsed configuration tree is created:

- Directives which are executed immediately rather than being stored in the parsed configuration are not listed. These include `ServerRoot`, `LoadModule`, and `LoadFile`.
- Directives which control the configuration file itself, such as `Include`, `<IfModule>` and `<IfDefine>` are not listed, but the included configuration directives are.
- Comments are not listed. (This may be considered a feature.)
- Configuration directives from `.htaccess` files are not listed (since they do not form part of the permanent server configuration).
- Container directives such as `<Directory>` are listed normally, but `mod_info` cannot figure out the line number for the closing `</Directory>`.
- Directives generated by third party modules such as `mod_perl` might not be listed.



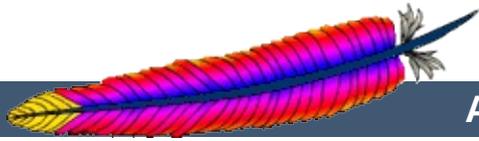
Description:	Adds additional information to the module information displayed by the server-info handler
Syntax:	<code>AddModuleInfo <i>module-name</i> <i>string</i></code>
Context:	server config, virtual host
Status:	Extension
Module:	mod_info
Compatibility:	Apache 1.3 and above

This allows the content of *string* to be shown as HTML interpreted, **Additional Information** for the module *module-name*. Example:

```
AddModuleInfo mod_deflate.c 'See <a \
href="http://www.apache.org/docs/2.2/mod/mod_deflate.html">\
http://www.apache.org/docs/2.2/mod/mod_deflate.html</a>'
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_isapi

Description:	ISAPI Extensions within Apache for Windows
Status:	Base
Module Identifier:	isapi_module
Source File:	mod_isapi.c
Compatibility:	Win32 only

Summary

This module implements the Internet Server extension API. It allows Internet Server extensions (e.g. ISAPI .dll modules) to be served by Apache for Windows, subject to the noted restrictions.

ISAPI extension modules (.dll files) are written by third parties. The Apache Group does not author these modules, so we provide no support for them. Please contact the ISAPI's author directly if you are experiencing problems running their ISAPI extension. **Please do not post such problems to Apache's lists or bug reporting pages.**



In the server configuration file, use the [AddHandler](#) directive to associate ISAPI files with the `isapi-handler` handler, and map it to them with their file extensions. To enable any `.dll` file to be processed as an ISAPI extension, edit the `httpd.conf` file and add the following line:

```
AddHandler isapi-handler .dll
```

In older versions of the Apache server, `isapi-isa` was the proper handler name, rather than `isapi-handler`. For compatibility, configurations may continue using `isapi-isa` through all versions of Apache prior to 2.3.0.

There is no capability within the Apache server to leave a requested module loaded. However, you may preload and keep a specific module loaded by using the following syntax in your `httpd.conf`:

```
ISAPICacheFile c:/WebWork/Scripts/ISAPI/mytest.dll
```

Whether or not you have preloaded an ISAPI extension, all ISAPI extensions are governed by the same permissions and restrictions as CGI scripts. That is, [Options](#) `ExecCGI` must be set for the directory that contains the ISAPI `.dll` file.

Review the [Additional Notes](#) and the [Programmer's Journal](#) for additional details and clarification of the specific ISAPI support offered by [mod_isapi](#).



Apache's ISAPI implementation conforms to all of the ISAPI 2.0 specification, except for some "Microsoft-specific" extensions dealing with asynchronous I/O. Apache's I/O model does not allow asynchronous reading and writing in a manner that the ISAPI could access. If an ISA tries to access unsupported features, including async I/O, a message is placed in the error log to help with debugging. Since these messages can become a flood, the directive `ISAPILogNotSupported Off` exists to quiet this noise.

Some servers, like Microsoft IIS, load the ISAPI extension into the server and keep it loaded until memory usage is too high, or unless configuration options are specified. Apache currently loads and unloads the ISAPI extension each time it is requested, unless the `ISAPICacheFile` directive is specified. This is inefficient, but Apache's memory model makes this the most effective method. Many ISAPI modules are subtly incompatible with the Apache server, and unloading these modules helps to ensure the stability of the server.

Also, remember that while Apache supports ISAPI Extensions, it **does not support ISAPI Filters**. Support for filters may be added at a later date, but no support is planned at this time.



If you are programming Apache 2.0 `mod_isapi` modules, you must limit your calls to `ServerSupportFunction` to the following directives:

HSE_REQ_SEND_URL_REDIRECT_RESP

Redirect the user to another location.
This must be a fully qualified URL (e.g. `http://server/location`).

HSE_REQ_SEND_URL

Redirect the user to another location.
This cannot be a fully qualified URL, you are not allowed to pass the protocol or a server name (e.g. simply `/location`).
This redirection is handled by the server, not the browser.

Warning

In their recent documentation, Microsoft appears to have abandoned the distinction between the two `HSE_REQ_SEND_URL` functions. Apache continues to treat them as two distinct functions with different requirements and behaviors.

HSE_REQ_SEND_RESPONSE_HEADER

Apache accepts a response body following the header if it follows the blank line (two consecutive newlines) in the headers string argument. This body cannot contain NULLs, since the headers argument is NULL terminated.

HSE_REQ_DONE_WITH_SESSION

Apache considers this a no-op, since the session will be finished when the ISAPI returns from processing.

HSE_REQ_MAP_URL_TO_PATH

Apache will translate a virtual name to a physical name.

HSE_APPEND_LOG_PARAMETER

This logged message may be captured in any of the following logs:

- in the `\"%{isapi-parameter}n\"` component in a [CustomLog](#) directive
- in the `%q` log component with the [ISAPIAppendLogToQuery](#) On directive
- in the error log with the [ISAPIAppendLogToErrors](#) On directive

The first option, the `%{isapi-parameter}n` component, is always available and preferred.

HSE_REQ_IS_KEEP_CONN

Will return the negotiated Keep-Alive status.

HSE_REQ_SEND_RESPONSE_HEADER_EX

Will behave as documented, although the `fKeepConn` flag is ignored.

HSE_REQ_IS_CONNECTED

Will report false if the request has been aborted.

Apache returns `FALSE` to any unsupported call to `ServerSupportFunction`, and sets the `GetLastError` value to `ERROR_INVALID_PARAMETER`.

`ReadClient` retrieves the request body exceeding the initial buffer (defined by [ISAPIReadAheadBuffer](#)). Based on the [ISAPIReadAheadBuffer](#) setting (number of bytes to buffer prior to calling the ISAPI handler) shorter requests are sent complete to the extension when it is invoked. If the request is longer, the ISAPI extension must use `ReadClient` to retrieve the remaining request body.

`WriteClient` is supported, but only with the `HSE_IO_SYNC` flag or no option flag (value of 0). Any other `WriteClient` request will be rejected with a return value of `FALSE`, and a `GetLastError` value of `ERROR_INVALID_PARAMETER`.

GetServerVariable is supported, although extended server variables do not exist (as defined by other servers.) All the usual Apache CGI environment variables are available from GetServerVariable, as well as the ALL_HTTP and ALL_RAW values.

Apache 2.0 [mod_isapi](#) supports additional features introduced in later versions of the ISAPI specification, as well as limited emulation of async I/O and the TransmitFile semantics. Apache also supports preloading ISAPI .dlls for performance, neither of which were not available under Apache 1.3 mod_isapi.



Description:	Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the error log
Syntax:	ISAPIAppendLogToErrors on off
Default:	ISAPIAppendLogToErrors off
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_isapi

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the server error log.



Description:	Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field
Syntax:	ISAPIAppendLogToQuery on off
Default:	ISAPIAppendLogToQuery on
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_isapi

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field (appended to the [CustomLog](#) %q component).



Description:	ISAPI .dll files to be loaded at startup
Syntax:	ISAPICacheFile <i>file-path</i> [<i>file-path</i>] ...
Context:	server config, virtual host
Status:	Base
Module:	mod_isapi

Specifies a space-separated list of file names to be loaded when the Apache server is launched, and remain loaded until the server is shut down. This directive may be repeated for every ISAPI .dll file desired. The full path name of each file should be specified. If the path name is not absolute, it will be treated relative to [ServerRoot](#).



Description:	Fake asynchronous support for ISAPI callbacks
Syntax:	ISAPIFakeAsync on off
Default:	ISAPIFakeAsync off
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_isapi

While set to on, asynchronous support for ISAPI callbacks is simulated.



Description:	Log unsupported feature requests from ISAPI extensions
Syntax:	ISAPILogNotSupported on off
Default:	ISAPILogNotSupported off
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_isapi

Logs all requests for unsupported features from ISAPI extensions in the server error log. This may help administrators to track down problems. Once set to on and all desired ISAPI modules are functioning, it should be set back to off.

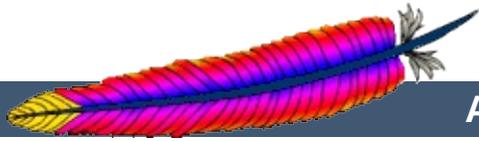


Description:	Size of the Read Ahead Buffer sent to ISAPI extensions
Syntax:	ISAPIReadAheadBuffer <i>size</i>
Default:	ISAPIReadAheadBuffer 49152
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_isapi

Defines the maximum size of the Read Ahead Buffer sent to ISAPI extensions when they are initially invoked. All remaining data must be retrieved using the `ReadClient` callback; some ISAPI extensions may not support the `ReadClient` function. Refer questions to the ISAPI extension's author.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_ldap

Description:	LDAP connection pooling and result caching services for use by other LDAP modules
Status:	Extension
Module Identifier:	ldap_module
Source File:	util_ldap.c
Compatibility:	Available in version 2.0.41 and later

Summary

This module was created to improve the performance of websites relying on backend connections to LDAP servers. In addition to the functions provided by the standard LDAP libraries, this module adds an LDAP connection pool and an LDAP shared memory cache.

To enable this module, LDAP support must be compiled into apr-util. This is achieved by adding the `--with-ldap` flag to the [configure](#) script when building Apache.

SSL/TLS support is dependent on which LDAP toolkit has been linked to [APR](#). As of this writing, APR-util supports: [OpenLDAP SDK](#) (2.x or later), [Novell LDAP SDK](#), [Mozilla LDAP SDK](#), native Solaris LDAP SDK (Mozilla based), native Microsoft LDAP SDK, or the [iPlanet \(Netscape\) SDK](#). See the [APR](#) website for details.



Example Configuration

The following is an example configuration that uses [mod_ldap](#) to increase the performance of HTTP Basic authentication provided by [mod_authnz_ldap](#).

```
# Enable the LDAP connection pool and shared
# memory cache. Enable the LDAP cache status
# handler. Requires that mod_ldap and mod_authnz_ldap
# be loaded. Change the "yourdomain.example.com" to
# match your domain.

LDAPSharedCacheSize 500000
LDAPCacheEntries 1024
LDAPCacheTTL 600
LDAPOpCacheEntries 1024
LDAPOpCacheTTL 600

<Location /ldap-status>
  SetHandler ldap-status
  Order deny,allow
  Deny from all
  Allow from yourdomain.example.com
  AuthLDAPURL ldap://127.0.0.1/dc=example,dc=com?uid?one
  AuthzLDAPAuthoritative off
  Require valid-user
</Location>
```



LDAP connections are pooled from request to request. This allows the LDAP server to remain connected and bound ready for the next request, without the need to unbind/connect/rebind. The performance advantages are similar to the effect of HTTP keepalives.

On a busy server it is possible that many requests will try and access the same LDAP server connection simultaneously. Where an LDAP connection is in use, Apache will create a new connection alongside the original one. This ensures that the connection pool does not become a bottleneck.

There is no need to manually enable connection pooling in the Apache configuration. Any module using this module for access to LDAP services will share the connection pool.



LDAP Caching

For improved performance, [mod_ldap](#) uses an aggressive caching strategy to minimize the number of times that the LDAP server must be contacted. Caching can easily double or triple the throughput of Apache when it is serving pages protected with `mod_authnz_ldap`. In addition, the load on the LDAP server will be significantly decreased.

[mod_ldap](#) supports two types of LDAP caching during the search/bind phase with a *search/bind cache* and during the compare phase with two *operation caches*. Each LDAP URL that is used by the server has its own set of these three caches.

The Search/Bind Cache

The process of doing a search and then a bind is the most time-consuming aspect of LDAP operation, especially if the directory is large. The search/bind cache is used to cache all searches that resulted in successful binds. Negative results (*i.e.*, unsuccessful searches, or searches that did not result in a successful bind) are not cached. The rationale behind this decision is that connections with invalid credentials are only a tiny percentage of the total number of connections, so by not caching invalid credentials, the size of the cache is reduced.

[mod_ldap](#) stores the username, the DN retrieved, the password used to bind, and the time of the bind in the cache. Whenever a new connection is initiated with the same username, [mod_ldap](#) compares the password of the new connection with the password in the cache. If the passwords match, and if the cached entry is not too old, [mod_ldap](#) bypasses the search/bind phase.

The search and bind cache is controlled with the [LDAPCacheEntries](#) and [LDAPCacheTTL](#) directives.

Operation Caches

During attribute and distinguished name comparison functions, [mod_ldap](#) uses two operation caches to cache the compare operations. The first compare cache is used to cache the results of compares done to test for LDAP group membership. The second compare cache is used to cache the results of comparisons done between distinguished names.

The behavior of both of these caches is controlled with the [LDAPOpCacheEntries](#) and [LDAPOpCacheTTL](#) directives.

Monitoring the Cache

[mod_ldap](#) has a content handler that allows administrators to monitor the cache performance. The name of the content handler is `ldap-status`, so the following directives could be used to access the [mod_ldap](#) cache information:

```
<Location /server/cache-info>  
  SetHandler ldap-status  
</Location>
```

By fetching the URL `http://servername/cache-info`, the administrator can get a status report of every cache that is used by [mod_ldap](#) cache. Note that if Apache does not support shared memory, then each [httpd](#) instance has its own cache, so reloading the URL will result in different information each time, depending on which [httpd](#) instance processes the request.



The ability to create an SSL and TLS connections to an LDAP server is defined by the directives [LDAPTrustedGlobalCert](#), [LDAPTrustedClientCert](#) and [LDAPTrustedMode](#). These directives specify the CA and optional client certificates to be used, as well as the type of encryption to be used on the connection (none, SSL or TLS/STARTTLS).

```
# Establish an SSL LDAP connection on port 636. Requires that
# mod_ldap and mod_authnz_ldap be loaded. Change the
# "yourdomain.example.com" to match your domain.
```

```
LDAPTrustedGlobalCert CA_DER /certs/certfile.der
```

```
<Location /ldap-status>
  SetHandler ldap-status
  Order deny,allow
  Deny from all
  Allow from yourdomain.example.com
  AuthLDAPURL ldaps://127.0.0.1/dc=example,dc=com?uid?one
  AuthzLDAPAuthoritative off
  Require valid-user
</Location>
```

```
# Establish a TLS LDAP connection on port 389. Requires that
# mod_ldap and mod_authnz_ldap be loaded. Change the
# "yourdomain.example.com" to match your domain.
```

```
LDAPTrustedGlobalCert CA_DER /certs/certfile.der
```

```
<Location /ldap-status>
  SetHandler ldap-status
  Order deny,allow
  Deny from all
  Allow from yourdomain.example.com
  AuthLDAPURL ldap://127.0.0.1/dc=example,dc=com?uid?one TLS
  AuthzLDAPAuthoritative off
  Require valid-user
</Location>
```



The different LDAP SDKs have widely different methods of setting and handling both CA and client side certificates.

If you intend to use SSL or TLS, read this section CAREFULLY so as to understand the differences between configurations on the different LDAP toolkits supported.

Netscape/Mozilla/iPlanet SDK

CA certificates are specified within a file called cert7.db. The SDK will not talk to any LDAP server whose certificate was not signed by a CA specified in this file. If client certificates are required, an optional key3.db file may be specified with an optional password. The secmod file can be specified if required. These files are in the same format as used by the Netscape Communicator or Mozilla web browsers. The easiest way to obtain these files is to grab them from your browser installation.

Client certificates are specified per connection using the LDAPTrustedClientCert directive by referring to the certificate "nickname". An optional password may be specified to unlock the certificate's private key.

The SDK supports SSL only. An attempt to use STARTTLS will cause an error when an attempt is made to contact the LDAP server at runtime.

```
# Specify a Netscape CA certificate file
LDAPTrustedGlobalCert CA_CERT7_DB /certs/cert7.db
# Specify an optional key3.db file for client certificate support
LDAPTrustedGlobalCert CERT_KEY3_DB /certs/key3.db
# Specify the secmod file if required
LDAPTrustedGlobalCert CA_SECMOD /certs/secmod
<Location /ldap-status>
  SetHandler ldap-status
  Order deny,allow
  Deny from all
  Allow from yourdomain.example.com
  LDAPTrustedClientCert CERT_NICKNAME <nickname> [password]
  AuthLDAPURL ldaps://127.0.0.1/dc=example,dc=com?uid?one
  AuthzLDAPAuthoritative off
```

```
Require valid-user
</Location>
```

Novell SDK

One or more CA certificates must be specified for the Novell SDK to work correctly. These certificates can be specified as binary DER or Base64 (PEM) encoded files.

Note: Client certificates are specified globally rather than per connection, and so must be specified with the LDAPTrustedGlobalCert directive as below. Trying to set client certificates via the LDAPTrustedClientCert directive will cause an error to be logged when an attempt is made to connect to the LDAP server..

The SDK supports both SSL and STARTTLS, set using the LDAPTrustedMode parameter. If an ldaps:// URL is specified, SSL mode is forced, override this directive.

```
# Specify two CA certificate files
LDAPTrustedGlobalCert CA_DER /certs/cacert1.der
LDAPTrustedGlobalCert CA_BASE64 /certs/cacert2.pem
# Specify a client certificate file and key
LDAPTrustedGlobalCert CERT_BASE64 /certs/cert1.pem
LDAPTrustedGlobalCert KEY_BASE64 /certs/key1.pem [password]
# Do not use this directive, as it will throw an error
#LDAPTrustedClientCert CERT_BASE64 /certs/cert1.pem
```

OpenLDAP SDK

One or more CA certificates must be specified for the OpenLDAP SDK to work correctly. These certificates can be specified as binary DER or Base64 (PEM) encoded files.

Client certificates are specified per connection using the LDAPTrustedClientCert directive.

The documentation for the SDK claims to support both SSL and

STARTTLS, however STARTTLS does not seem to work on all versions of the SDK. The SSL/TLS mode can be set using the LDAPTrustedMode parameter. If an ldaps:// URL is specified, SSL mode is forced. The OpenLDAP documentation notes that SSL (ldaps://) support has been deprecated to be replaced with TLS, although the SSL functionality still works.

```
# Specify two CA certificate files
LDAPTrustedGlobalCert CA_DER /certs/cacert1.der
LDAPTrustedGlobalCert CA_BASE64 /certs/cacert2.pem
<Location /ldap-status>
  SetHandler ldap-status
  Order deny,allow
  Deny from all
  Allow from yourdomain.example.com
  LDAPTrustedClientCert CERT_BASE64 /certs/cert1.pem
  LDAPTrustedClientCert KEY_BASE64 /certs/key1.pem
  AuthLDAPURL ldaps://127.0.0.1/dc=example,dc=com?uid?one
  AuthzLDAPAuthoritative off
  Require valid-user
</Location>
```

Solaris SDK

SSL/TLS for the native Solaris LDAP libraries is not yet supported. If required, install and use the OpenLDAP libraries instead.

Microsoft SDK

SSL/TLS certificate configuration for the native Microsoft LDAP libraries is done inside the system registry, and no configuration directives are required.

Both SSL and TLS are supported by using the ldaps:// URL format, or by using the LDAPTrustedMode directive accordingly.

Note: The status of support for client certificates is not yet known for this toolkit.



Description:	Maximum number of entries in the primary LDAP cache
Syntax:	LDAPCacheEntries <i>number</i>
Default:	LDAPCacheEntries 1024
Context:	server config
Status:	Extension
Module:	mod_ldap

Specifies the maximum size of the primary LDAP cache. This cache contains successful search/binds. Set it to 0 to turn off search/bind caching. The default size is 1024 cached searches.



Description:	Time that cached items remain valid
Syntax:	LDAPCacheTTL <i>seconds</i>
Default:	LDAPCacheTTL 600
Context:	server config
Status:	Extension
Module:	mod_ldap

Specifies the time (in seconds) that an item in the search/bind cache remains valid. The default is 600 seconds (10 minutes).



Description:	Specifies the socket connection timeout in seconds
Syntax:	<code>LDAPConnectionTimeout</code> <i>seconds</i>
Context:	server config
Status:	Extension
Module:	<code>mod_ldap</code>

This directive configures the `LDAP_OPT_NETWORK_TIMEOUT` option in the underlying LDAP client library, when available. This value typically controls how long the LDAP client library will wait for the TCP connection to the LDAP server to complete.

If a connection is not successful with the timeout period, either an error will be returned or the LDAP client library will attempt to connect to a secondary LDAP server if one is specified (via a space-separated list of hostnames in the [AuthLDAPURL](#)).

The default is 10 seconds, if the LDAP client library linked with the server supports the `LDAP_OPT_NETWORK_TIMEOUT` option.

`LDAPConnectionTimeout` is only available when the LDAP client library linked with the server supports the `LDAP_OPT_NETWORK_TIMEOUT` option, and the ultimate behavior is dictated entirely by the LDAP client library.



Description:	Number of entries used to cache LDAP compare operations
Syntax:	LDAPOpCacheEntries <i>number</i>
Default:	LDAPOpCacheEntries 1024
Context:	server config
Status:	Extension
Module:	mod_ldap

This specifies the number of entries `mod_ldap` will use to cache LDAP compare operations. The default is 1024 entries. Setting it to 0 disables operation caching.



Description:	Time that entries in the operation cache remain valid
Syntax:	LDAPOpCacheTTL <i>seconds</i>
Default:	LDAPOpCacheTTL 600
Context:	server config
Status:	Extension
Module:	mod_ldap

Specifies the time (in seconds) that entries in the operation cache remain valid. The default is 600 seconds.



Description:	Sets the shared memory cache file
Syntax:	LDAPSharedCacheFile <i>directory-path/filename</i>
Context:	server config
Status:	Extension
Module:	mod_ldap

Specifies the directory path and file name of the shared memory cache file. If not set, anonymous shared memory will be used if the platform supports it.



Description:	Size in bytes of the shared-memory cache
Syntax:	LDAPSharedCacheSize <i>bytes</i>
Default:	LDAPSharedCacheSize 500000
Context:	server config
Status:	Extension
Module:	mod_ldap

Specifies the number of bytes to allocate for the shared memory cache. The default is 500kb. If set to 0, shared memory caching will not be used.



Description:	Sets the file containing or nickname referring to a per connection client certificate. Not all LDAP toolkits support per connection client certificates.
Syntax:	<code>LDAPTrustedClientCert <i>type</i> <i>directory-path/filename/nickname</i> [<i>password</i>]</code>
Context:	server config, virtual host, directory, .htaccess
Status:	Extension
Module:	mod_ldap

It specifies the directory path, file name or nickname of a per connection client certificate used when establishing an SSL or TLS connection to an LDAP server. Different locations or directories may have their own independent client certificate settings. Some LDAP toolkits (notably Novell) do not support per connection client certificates, and will throw an error on LDAP server connection if you try to use this directive (Use the LDAPTrustedGlobalCert directive instead for Novell client certificates - See the SSL/TLS certificate guide above for details). The type specifies the kind of certificate parameter being set, depending on the LDAP toolkit being used. Supported types are:

- CERT_DER - binary DER encoded client certificate
- CERT_BASE64 - PEM encoded client certificate
- CERT_NICKNAME - Client certificate "nickname" (Netscape SDK)
- KEY_DER - binary DER encoded private key
- KEY_BASE64 - PEM encoded private key



Description:	Sets the file or database containing global trusted Certificate Authority or global client certificates
Syntax:	<code>LDAPTrustedGlobalCert <i>type</i> <i>directory-path/filename</i> [<i>password</i>]</code>
Context:	server config
Status:	Extension
Module:	mod_ldap

It specifies the directory path and file name of the trusted CA certificates and/or system wide client certificates `mod_ldap` should use when establishing an SSL or TLS connection to an LDAP server. Note that all certificate information specified using this directive is applied globally to the entire server installation. Some LDAP toolkits (notably Novell) require all client certificates to be set globally using this directive. Most other toolkits require clients certificates to be set per Directory or per Location using `LDAPTrustedClientCert`. If you get this wrong, an error may be logged when an attempt is made to contact the LDAP server, or the connection may silently fail (See the SSL/TLS certificate guide above for details). The type specifies the kind of certificate parameter being set, depending on the LDAP toolkit being used. Supported types are:

- CA_DER - binary DER encoded CA certificate
- CA_BASE64 - PEM encoded CA certificate
- CA_CERT7_DB - Netscape cert7.db CA certificate database file
- CA_SECMOD - Netscape secmod database file
- CERT_DER - binary DER encoded client certificate
- CERT_BASE64 - PEM encoded client certificate
- CERT_KEY3_DB - Netscape key3.db client certificate database file
- CERT_NICKNAME - Client certificate "nickname" (Netscape SDK)
- CERT_PFX - PKCS#12 encoded client certificate (Novell SDK)
- KEY_DER - binary DER encoded private key
- KEY_BASE64 - PEM encoded private key
- KEY_PFX - PKCS#12 encoded private key (Novell SDK)



Description:	Specifies the SSL/TLS mode to be used when connecting to an LDAP server.
Syntax:	LDAPTrustedMode <i>type</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ldap

The following modes are supported:

- NONE - no encryption
- SSL - ldaps:// encryption on default port 636
- TLS - STARTTLS encryption on default port 389

Not all LDAP toolkits support all the above modes. An error message will be logged at runtime if a mode is not supported, and the connection to the LDAP server will fail.

If an ldaps:// URL is specified, the mode becomes SSL and the setting of LDAPTrustedMode is ignored.

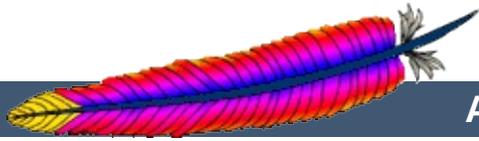


Description:	Force server certificate verification
Syntax:	LDAPVerifyServerCert <i>On Off</i>
Default:	LDAPVerifyServerCert On
Context:	server config
Status:	Extension
Module:	mod_ldap

Specifies whether to force the verification of a server certificate when establishing an SSL connection to the LDAP server.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_log_config`

Description:	Logging of the requests made to the server
Status:	Base
Module Identifier:	<code>log_config_module</code>
Source File:	<code>mod_log_config.c</code>

Summary

This module provides for flexible logging of client requests. Logs are written in a customizable format, and may be written directly to a file, or to an external program. Conditional logging is provided so that individual requests may be included or excluded from the logs based on characteristics of the request.

Three directives are provided by this module: [TransferLog](#) to create a log file, [LogFormat](#) to set a custom format, and [CustomLog](#) to define a log file and format in one step. The [TransferLog](#) and [CustomLog](#) directives can be used multiple times in each server to cause each request to be logged to multiple files.

See also

[Apache Log Files](#)



Custom Log Format

The format argument to the [LogFormat](#) and [CustomLog](#) directives is a string. This string is used to log each request to the log file. It can contain literal characters copied into the log files and the C-style control characters "\n" and "\t" to represent new-lines and tabs. Literal quotes and backslashes should be escaped with backslashes.

The characteristics of the request itself are logged by placing "%" directives in the format string, which are replaced in the log file by the values as follows:

Format String	Description
%%	The percent sign
%a	Remote IP-address
%A	Local IP-address
%B	Size of response in bytes, excluding HTTP headers.
%b	Size of response in bytes, excluding HTTP headers. In CLF format, <i>i.e.</i> a '-' rather than a 0 when no bytes are sent.
%{ <i>Foo</i> bar}C	The contents of cookie <i>Foo</i> bar in the request sent to the server. Only version 0 cookies are fully supported.
%D	The time taken to serve the request, in microseconds.
%{ <i>FOO</i> BAR}e	The contents of the environment variable <i>FOO</i> BAR
%f	Filename
%h	Remote host
%H	The request protocol
%{ <i>Foo</i> bar}i	The contents of <i>Foo</i> bar: header line(s) in the request sent to the server. Changes made by other modules (e.g. mod_headers) affect this. If you're interested in what the request header was prior to when most modules would have modified it, use mod_setenvif to copy the header into an internal environment

	variable and log that value with the <code>%{VARNAME}e</code> described above.
<code>%k</code>	Number of keepalive requests handled on this connection. Interesting if KeepAlive is being used, so that, for example, a '1' means the first keepalive request after the initial one, '2' the second, etc...; otherwise this is always 0 (indicating the initial request). Available in versions 2.2.11 and later.
<code>%l</code>	Remote logname (from <code>identd</code> , if supplied). This will return a dash unless <code>mod_ident</code> is present and IdentityCheck is set On.
<code>%m</code>	The request method
<code>%{Foobar}n</code>	The contents of note <i>Foobar</i> from another module.
<code>%{Foobar}o</code>	The contents of <i>Foobar</i> : header line(s) in the reply.
<code>%p</code>	The canonical port of the server serving the request
<code>%{format}p</code>	The canonical port of the server serving the request or the server's actual port or the client's actual port. Valid formats are <code>canonical</code> , <code>local</code> , or <code>remote</code> .
<code>%P</code>	The process ID of the child that serviced the request.
<code>%{format}P</code>	The process ID or thread id of the child that serviced the request. Valid formats are <code>pid</code> , <code>tid</code> , and <code>hextid</code> . <code>hextid</code> requires APR 1.2.0 or higher.
<code>%q</code>	The query string (prepended with a ? if a query string exists, otherwise an empty string)
<code>%r</code>	First line of request
<code>%R</code>	The handler generating the response (if any).
<code>%s</code>	Status. For requests that got internally redirected, this is the status of the <i>*original*</i> request --- <code>%>s</code> for the last.
<code>%t</code>	Time the request was received (standard english format)

<p><code>%{format}t</code></p>	<p>The time, in the form given by <code>format</code>, which should be in an extended <code>strftime(3)</code> format (potentially localized). If the format starts with <code>begin:</code> (default) the time is taken at the beginning of the request processing. If it starts with <code>end:</code> it is the time when the log entry gets written, close to the end of the request processing. In addition to the formats supported by <code>strftime(3)</code>, the following format tokens are supported:</p> <table border="1" data-bbox="492 695 1419 1035"> <tr> <td><code>sec</code></td> <td>number of seconds since the Epoch</td> </tr> <tr> <td><code>msec</code></td> <td>number of milliseconds since the Epoch</td> </tr> <tr> <td><code>usec</code></td> <td>number of microseconds since the Epoch</td> </tr> <tr> <td><code>msec_frac</code></td> <td>millisecond fraction</td> </tr> <tr> <td><code>usec_frac</code></td> <td>microsecond fraction</td> </tr> </table> <p>These tokens can not be combined with each other or <code>strftime(3)</code> formatting in the same format string. You can use multiple <code>%{format}t</code> tokens instead. The extended <code>strftime(3)</code> tokens are available in 2.2.30 and later.</p>	<code>sec</code>	number of seconds since the Epoch	<code>msec</code>	number of milliseconds since the Epoch	<code>usec</code>	number of microseconds since the Epoch	<code>msec_frac</code>	millisecond fraction	<code>usec_frac</code>	microsecond fraction
<code>sec</code>	number of seconds since the Epoch										
<code>msec</code>	number of milliseconds since the Epoch										
<code>usec</code>	number of microseconds since the Epoch										
<code>msec_frac</code>	millisecond fraction										
<code>usec_frac</code>	microsecond fraction										
<p><code>%T</code></p>	<p>The time taken to serve the request, in seconds.</p>										
<p><code>%{UNIT}T</code></p>	<p>The time taken to serve the request, in a time unit given by <code>UNIT</code>. Valid units are <code>ms</code> for milliseconds, <code>us</code> for microseconds, and <code>s</code> for seconds. Using <code>s</code> gives the same result as <code>%T</code> without any format; using <code>us</code> gives the same result as <code>%D</code>. Combining <code>%T</code> with a unit is available in 2.2.30 and later.</p>										
<p><code>%u</code></p>	<p>Remote user (from <code>auth</code>; may be bogus if return status (<code>%s</code>) is 401)</p>										
<p><code>%U</code></p>	<p>The URL path requested, not including any query</p>										

	string.												
%v	The canonical ServerName of the server serving the request.												
%V	The server name according to the UseCanonicalName setting.												
%X	<p>Connection status when response is completed:</p> <table border="1"> <tr> <td>X</td> <td>connection aborted before the response</td> </tr> <tr> <td>=</td> <td>completed.</td> </tr> <tr> <td>+</td> <td>connection may be kept alive after the response</td> </tr> <tr> <td>=</td> <td>is sent.</td> </tr> <tr> <td>-</td> <td>connection will be closed after the response is</td> </tr> <tr> <td>=</td> <td>sent.</td> </tr> </table> <p>(This directive was %c in late versions of Apache 1.3, but this conflicted with the historical ssl %{var}c syntax.)</p>	X	connection aborted before the response	=	completed.	+	connection may be kept alive after the response	=	is sent.	-	connection will be closed after the response is	=	sent.
X	connection aborted before the response												
=	completed.												
+	connection may be kept alive after the response												
=	is sent.												
-	connection will be closed after the response is												
=	sent.												
%I	Bytes received, including request and headers, cannot be zero. You need to enable mod_logio to use this.												
%O	Bytes sent, including headers, cannot be zero. You need to enable mod_logio to use this.												
% {VARNAME}^ti	The contents of <i>VARNAME</i> : trailer line(s) in the request sent to the server.												
% {VARNAME}^to	The contents of <i>VARNAME</i> : trailer line(s) in the response sent from the server.												

Modifiers

Particular items can be restricted to print only for responses with specific HTTP status codes by placing a comma-separated list of status codes immediately following the "%". For example, "%400, 501{User-agent}i" logs User-agent on 400 errors and 501 errors only. For

other status codes, the literal string "-" will be logged. The status code list may be preceded by a "!" to indicate negation:

"%!200,304,302{Referer}i" logs Referer on all requests that do *not* return one of the three specified codes.

The modifiers "<" and ">" can be used for requests that have been internally redirected to choose whether the original or final (respectively) request should be consulted. By default, the % directives %s, %U, %T, %D, and %r look at the original request while all others look at the final request. So for example, %>s can be used to record the final status of the request and %<u can be used to record the original authenticated user on a request that is internally redirected to an unauthenticated resource.

Some Notes

For security reasons, starting with version 2.0.46, non-printable and other special characters in %r, %i and %o are escaped using \xhh sequences, where hh stands for the hexadecimal representation of the raw byte. Exceptions from this rule are " and \, which are escaped by prepending a backslash, and all whitespace characters, which are written in their C-style notation (\n, \t, etc). In versions prior to 2.0.46, no escaping was performed on these strings so you had to be quite careful when dealing with raw log files.

In httpd 2.0, unlike 1.3, the %b and %B format strings do not represent the number of bytes sent to the client, but simply the size in bytes of the HTTP response (which will differ, for instance, if the connection is aborted, or if SSL is used). The %O format provided by [mod_logio](#) will log the actual number of bytes sent over the network.

Note: [mod_cache](#) is implemented as a quick-handler and not as a standard handler. Therefore, the %R format string will not return any handler information when content caching is involved.

Examples

Some commonly used log format strings are:

Common Log Format (CLF)

```
"%h %l %u %t \"%r\" %>s %b"
```

Common Log Format with Virtual Host

```
"%v %h %l %u %t \"%r\" %>s %b"
```

NCSA extended/combined log format

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
```

Referer log format

```
"%{Referer}i -> %U"
```

Agent (Browser) log format

```
"%{User-agent}i"
```

You can use the `{format}t` directive multiple times to build up a time format using the extended format tokens like `msec_frac`:

Timestamp including milliseconds

```
"%{%d/%b/%Y %T}t.%{msec_frac}t %{%Z}t"
```



See the [security tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.



Description:	Buffer log entries in memory before writing to disk
Syntax:	BufferedLogs On Off
Default:	BufferedLogs Off
Context:	server config
Status:	Base
Module:	mod_log_config
Compatibility:	Available in versions 2.0.41 and later.

The **BufferedLogs** directive causes [mod_log_config](#) to store several log entries in memory and write them together to disk, rather than writing them after each request. On some systems, this may result in more efficient disk access and hence higher performance. It may be set only once for the entire server; it cannot be configured per virtual-host.

This directive is experimental and should be used with caution.



Description:	Sets filename for the logging of cookies
Syntax:	CookieLog <i>filename</i>
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config
Compatibility:	This directive is deprecated.

The **CookieLog** directive sets the filename for logging of cookies. The filename is relative to the **ServerRoot**. This directive is included only for compatibility with `mod_cookies`, and is deprecated.



Description:	Sets filename and format of log file
Syntax:	<code>CustomLog <i>file</i> <i>pipe</i> <i>format</i> <i>nickname</i> [env=[!]<i>environment-variable</i>]</code>
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config

The `CustomLog` directive is used to log requests to the server. A log format is specified, and the logging can optionally be made conditional on request characteristics using environment variables.

The first argument, which specifies the location to which the logs will be written, can take one of the following two types of values:

file

A filename, relative to the [ServerRoot](#).

pipe

The pipe character "|", followed by the path to a program to receive the log information on its standard input. See the notes on [piped logs](#) for more information.

Security:

If a program is used, then it will be run as the user who started [httpd](#). This will be root if the server was started by root; be sure that the program is secure.

Note

When entering a file path on non-Unix platforms, care should be taken to make sure that only forward slashes are used even though the platform may allow the use of back slashes. In general it is a good idea to always use forward slashes throughout the

configuration files.

The second argument specifies what will be written to the log file. It can specify either a *nickname* defined by a previous [LogFormat](#) directive, or it can be an explicit *format* string as described in the [log formats](#) section.

For example, the following two sets of directives have exactly the same effect:

```
# CustomLog with format nickname
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common

# CustomLog with explicit format string
CustomLog logs/access_log "%h %l %u %t \"%r\" %>s %b"
```

The third argument is optional and controls whether or not to log a particular request based on the presence or absence of a particular variable in the server environment. If the specified [environment variable](#) is set for the request (or is not set, in the case of a 'env=!name' clause), then the request will be logged.

Environment variables can be set on a per-request basis using the [mod_setenvif](#) and/or [mod_rewrite](#) modules. For example, if you want to record requests for all GIF images on your server in a separate logfile but not in your main log, you can use:

```
SetEnvIf Request_URI \.gif$ gif-image
CustomLog gif-requests.log common env=gif-image
CustomLog nongif-requests.log common env=!gif-image
```

Or, to reproduce the behavior of the old `RefererIgnore` directive, you might use the following:

```
SetEnvIf Referer example\.com localreferer
CustomLog referer.log referer env=!localreferer
```



LogFormat Directive

Description:	Describes a format for use in a log file
Syntax:	LogFormat <i>format</i> [<i>nickname</i>] [<i>nickname</i>]
Default:	LogFormat "%h %l %u %t \"%r\" %>s %b"
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config

This directive specifies the format of the access log file.

The **LogFormat** directive can take one of two forms. In the first form, where only one argument is specified, this directive sets the log format which will be used by logs specified in subsequent **TransferLog** directives. The single argument can specify an explicit *format* as discussed in the [custom log formats](#) section above. Alternatively, it can use a *nickname* to refer to a log format defined in a previous **LogFormat** directive as described below.

The second form of the **LogFormat** directive associates an explicit *format* with a *nickname*. This *nickname* can then be used in subsequent **LogFormat** or **CustomLog** directives rather than repeating the entire format string. A **LogFormat** directive that defines a nickname **does nothing else** -- that is, it *only* defines the nickname, it doesn't actually apply the format and make it the default. Therefore, it will not affect subsequent **TransferLog** directives. In addition, **LogFormat** cannot use one nickname to define another nickname. Note that the nickname should not contain percent signs (%).

Example

```
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost_common
```



Description:	Specify location of a log file
Syntax:	TransferLog <i>file pipe</i>
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config

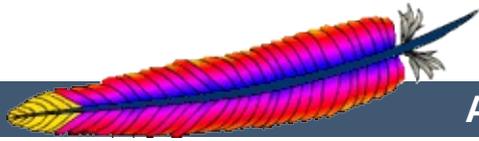
This directive has exactly the same arguments and effect as the [CustomLog](#) directive, with the exception that it does not allow the log format to be specified explicitly or for conditional logging of requests. Instead, the log format is determined by the most recently specified [LogFormat](#) directive which does not define a nickname. Common Log Format is used if no other format has been specified.

Example

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""  
TransferLog logs/access_log
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_log_forensic`

Description:	Forensic Logging of the requests made to the server
Status:	Extension
Module Identifier:	<code>log_forensic_module</code>
Source File:	<code>mod_log_forensic.c</code>
Compatibility:	<code>mod_unique_id</code> is no longer required since version 2.1

Summary

This module provides for forensic logging of client requests. Logging is done before and after processing a request, so the forensic log contains two log lines for each request. The forensic logger is very strict, which means:

- The format is fixed. You cannot modify the logging format at runtime.
- If it cannot write its data, the child process exits immediately and may dump core (depending on your [CoreDumpDirectory](#) configuration).

The `check_forensic` script, which can be found in the distribution's support directory, may be helpful in evaluating the forensic log output.

See also

[Apache Log Files](#)
[mod_log_config](#)



Forensic Log Format

Each request is logged two times. The first time is *before* it's processed further (that is, after receiving the headers). The second log entry is written *after* the request processing at the same time where normal logging occurs.

In order to identify each request, a unique request ID is assigned. This forensic ID can be cross logged in the normal transfer log using the %
{forensic-id}n format string. If you're using [mod_unique_id](#), its generated ID will be used.

The first line logs the forensic ID, the request line and all received headers, separated by pipe characters (|). A sample line looks like the following (all on one line):

```
+yQtJf8CoAB4AAFNBIEAAAAA|GET /manual/de/images/down.gif  
HTTP/1.1|Host:localhost%3a8080|User-Agent:Mozilla/5.0 (X11; U; Linux  
i686; en-US; rv%3a1.6) Gecko/20040216 Firefox/0.8|Accept:image/png,  
etc...
```

The plus character at the beginning indicates that this is the first log line of this request. The second line just contains a minus character and the ID again:

```
-yQtJf8CoAB4AAFNBIEAAAAA
```

The `check_forensic` script takes as its argument the name of the logfile. It looks for those +/- ID pairs and complains if a request was not completed.



See the [security tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

The log files may contain sensitive data such as the contents of `Authorization:` headers (which can contain passwords), so they should not be readable by anyone except the user that starts the server.



Description:	Sets filename of the forensic log
Syntax:	ForensicLog <i>filename pipe</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_log_forensic

The **ForensicLog** directive is used to log requests to the server for forensic analysis. Each log entry is assigned a unique ID which can be associated with the request using the normal **CustomLog** directive. **mod_log_forensic** creates a token called forensic-id, which can be added to the transfer log using the `%{forensic-id}n` format string.

The argument, which specifies the location to which the logs will be written, can take one of the following two types of values:

filename

A filename, relative to the **ServerRoot**.

pipe

The pipe character "|", followed by the path to a program to receive the log information on its standard input. The program name can be specified relative to the **ServerRoot** directive.

Security:

If a program is used, then it will be run as the user who started **httpd**. This will be root if the server was started by root; be sure that the program is secure or switches to a less privileged user.

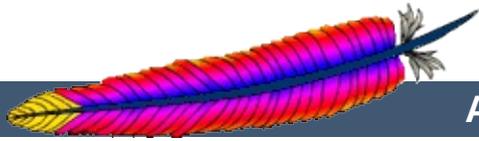
Note

When entering a file path on non-Unix platforms, care should be taken to make sure that only forward slashes are used even though the platform may allow the use of back slashes. In general

it is a good idea to always use forward slashes throughout the configuration files.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_logio

Description:	Logging of input and output bytes per request
Status:	Extension
Module Identifier:	logio_module
Source File:	mod_logio.c

Summary

This module provides the logging of input and output number of bytes received/sent per request. The numbers reflect the actual bytes as received on the network, which then takes into account the headers and bodies of requests and responses. The counting is done before SSL/TLS on input and after SSL/TLS on output, so the numbers will correctly reflect any changes made by encryption.

This module requires [mod_log_config](#).

When KeepAlive connections are used with SSL, the overhead of the SSL handshake is reflected in the byte count of the first request on the connection. When per-directory SSL renegotiation occurs, the bytes are associated with the request that triggered the renegotiation.

See also

[mod_log_config](#)

[Apache Log Files](#)



Custom Log Format

This module adds two new logging directives. The characteristics of the request itself are logged by placing "%" directives in the format string, which are replaced in the log file by the values as follows:

Format String	Description
%. . . I	Bytes received, including request and headers, cannot be zero.
%. . . O	Bytes sent, including headers, cannot be zero.

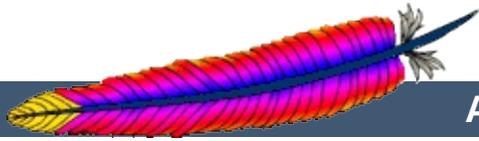
Usually, the functionality is used like this:

Combined I/O log format:

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\" %I %O"
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_mem_cache`

Description:	Content cache keyed to URIs
Status:	Extension
Module Identifier:	<code>mem_cache_module</code>
Source File:	<code>mod_mem_cache.c</code>

Summary

This module *requires* the service of `mod_cache`. It acts as a support module for `mod_cache` and provides a memory based storage manager. `mod_mem_cache` can be configured to operate in two modes: caching open file descriptors or caching objects in heap storage. `mod_mem_cache` is most useful when used to cache locally generated content or to cache backend server content for `mod_proxy` configured for `ProxyPass` (aka *reverse proxy*).

Content is stored in and retrieved from the cache using URI based keys. Content with access protection is not cached.

Note

In most cases `mod_disk_cache` should be the preferred choice. This is explained further in the [Caching Guide](#). In particular, this module's cache is *per-process*, which can be partially mitigated by configuring threaded MPMs to use fewer child processes via configuration of larger values for `ThreadsPerChild`. This module's cache is also limited to storing a single variant (see HTTP Vary: header) of each resource in the cache.

See also

[mod_cache](#)

mod disk cache



Description:	The maximum number of objects allowed to be placed in the cache
Syntax:	<code>MCacheMaxObjectCount <i>value</i></code>
Default:	<code>MCacheMaxObjectCount 1009</code>
Context:	server config
Status:	Extension
Module:	<code>mod_mem_cache</code>

The `MCacheMaxObjectCount` directive sets the maximum number of objects to be cached. The value is used to create the open hash table. If a new object needs to be inserted in the cache and the maximum number of objects has been reached, an object will be removed to allow the new object to be cached. The object to be removed is selected using the algorithm specified by `MCacheRemovalAlgorithm`.

Example

```
MCacheMaxObjectCount 13001
```



Description:	The maximum size (in bytes) of a document allowed in the cache
Syntax:	<code>MCacheMaxObjectSize</code> <i>bytes</i>
Default:	<code>MCacheMaxObjectSize</code> 10000
Context:	server config
Status:	Extension
Module:	<code>mod_mem_cache</code>

The `MCacheMaxObjectSize` directive sets the maximum allowable size, in bytes, of a document for it to be considered cacheable.

Example

```
MCacheMaxObjectSize 6400000
```

Note

The value of `MCacheMaxObjectSize` must be greater than the value specified by the `MCacheMinObjectSize` directive.



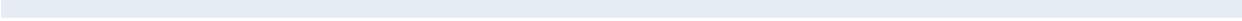
Description:	Maximum amount of a streamed response to buffer in memory before declaring the response uncacheable
Syntax:	<code>MCacheMaxStreamingBuffer <i>size_in_bytes</i></code>
Default:	<code>MCacheMaxStreamingBuffer</code> the smaller of 100000 or <code>MCacheMaxObjectSize</code>
Context:	server config
Status:	Extension
Module:	<code>mod_mem_cache</code>

The `MCacheMaxStreamingBuffer` directive specifies the maximum number of bytes of a streamed response to buffer before deciding that the response is too big to cache. A streamed response is one in which the entire content is not immediately available and in which the `Content-Length` may not be known. Sources of streaming responses include proxied responses and the output of CGI scripts. By default, a streamed response will *not* be cached unless it has a `Content-Length` header. The reason for this is to avoid using a large amount of memory to buffer a partial response that might end up being too large to fit in the cache. The `MCacheMaxStreamingBuffer` directive allows buffering of streamed responses that don't contain a `Content-Length` up to the specified maximum amount of space. If the maximum buffer space is reached, the buffered content is discarded and the attempt to cache is abandoned.

Note:

Using a nonzero value for `MCacheMaxStreamingBuffer` will not delay the transmission of the response to the client. As soon as `mod_mem_cache` copies a block of streamed content into a buffer, it sends the block on to the next output filter for delivery to the client.

```
# Enable caching of streamed responses up to 64KB:
MCacheMaxStreamingBuffer 65536
```



Description:	The minimum size (in bytes) of a document to be allowed in the cache
Syntax:	<code>MCacheMinObjectSize</code> <i>bytes</i>
Default:	<code>MCacheMinObjectSize</code> 1
Context:	server config
Status:	Extension
Module:	<code>mod_mem_cache</code>

The `MCacheMinObjectSize` directive sets the minimum size in bytes of a document for it to be considered cacheable.

Example

```
MCacheMinObjectSize 10000
```



Description:	The algorithm used to select documents for removal from the cache
Syntax:	<code>MCacheRemovalAlgorithm LRU GDSF</code>
Default:	<code>MCacheRemovalAlgorithm GDSF</code>
Context:	server config
Status:	Extension
Module:	<code>mod_mem_cache</code>

The `MCacheRemovalAlgorithm` directive specifies the algorithm used to select documents for removal from the cache. Two choices are available:

LRU (Least Recently Used)

LRU removes the documents that have not been accessed for the longest time.

GDSF (GreedyDual-Size)

GDSF assigns a priority to cached documents based on the cost of a cache miss and the size of the document. Documents with the lowest priority are removed first.

Example

```
MCacheRemovalAlgorithm GDSF
MCacheRemovalAlgorithm LRU
```



Description:	The maximum amount of memory used by the cache in KBytes
Syntax:	<code>MCacheSize</code> <i>KBytes</i>
Default:	<code>MCacheSize</code> 100
Context:	server config
Status:	Extension
Module:	<code>mod_mem_cache</code>

The `MCacheSize` directive sets the maximum amount of memory to be used by the cache, in KBytes (1024-byte units). If a new object needs to be inserted in the cache and the size of the object is greater than the remaining memory, objects will be removed until the new object can be cached. The object to be removed is selected using the algorithm specified by `MCacheRemovalAlgorithm`.

Example

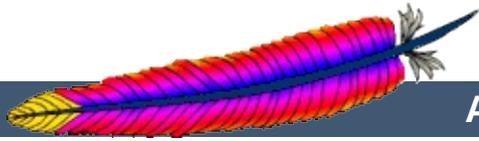
```
MCacheSize 700000
```

Note

The `MCacheSize` value must be greater than the value specified by the `MCacheMaxObjectSize` directive.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_mime

Description:	Associates the requested filename's extensions with the file's behavior (handlers and filters) and content (mime-type, language, character set and encoding)
Status:	Base
Module Identifier:	mime_module
Source File:	mod_mime.c

Summary

This module is used to associate various bits of "meta information" with files by their filename extensions. This information relates the filename of the document to its mime-type, language, character set and encoding. This information is sent to the browser, and participates in content negotiation, so the user's preferences are respected when choosing one of several possible files to serve. See [mod_negotiation](#) for more information about [content negotiation](#).

The directives [AddCharset](#), [AddEncoding](#), [AddLanguage](#) and [AddType](#) are all used to map file extensions onto the meta-information for that file. Respectively they set the character set, content-encoding, content-language, and [MIME-type](#) (content-type) of documents. The directive [TypesConfig](#) is used to specify a file which also maps extensions onto MIME types.

In addition, [mod_mime](#) may define the [handler](#) and [filters](#) that originate and process content. The directives [AddHandler](#), [AddOutputFilter](#), and [AddInputFilter](#) control the modules or scripts that serve the document. The [MultiviewsMatch](#) directive allows [mod_negotiation](#) to consider these file extensions to be included when testing Multiviews matches.

While [mod_mime](#) associates meta-information with filename extensions, the [core](#) server provides directives that are used to associate all the files in a given container (e.g., [<Location>](#), [<Directory>](#), or [<Files>](#)) with particular meta-information. These directives include [ForceType](#), [SetHandler](#), [SetInputFilter](#), and [SetOutputFilter](#). The core directives override any filename extension mappings defined in [mod_mime](#).

Note that changing the meta-information for a file does not change the value of the Last-Modified header. Thus, previously cached copies may still be used by a client or proxy, with the previous headers. If you change the meta-information (language, content type, character set or encoding) you may need to 'touch' affected files (updating their last modified date) to ensure that all visitors are receive the corrected content headers.

See also

- [MimeMagicFile](#)
- [AddDefaultCharset](#)
- [ForceType](#)
- [DefaultType](#)
- [SetHandler](#)
- [SetInputFilter](#)
- [SetOutputFilter](#)



Files can have more than one extension, and the order of the extensions is *normally* irrelevant. For example, if the file `welcome.html.fr` maps onto content type `text/html` and language French then the file `welcome.fr.html` will map onto exactly the same information. If more than one extension is given that maps onto the same type of meta-information, then the one to the right will be used, except for languages and content encodings. For example, if `.gif` maps to the [MIME-type](#) `image/gif` and `.html` maps to the MIME-type `text/html`, then the file `welcome.gif.html` will be associated with the MIME-type `text/html`.

[Languages](#) and [content encodings](#) are treated accumulative, because one can assign more than one language or encoding to a particular resource. For example, the file `welcome.html.en.de` will be delivered with `Content-Language: en, de` and `Content-Type: text/html`.

Care should be taken when a file with multiple extensions gets associated with both a [MIME-type](#) and a handler. This will usually result in the request being handled by the module associated with the handler. For example, if the `.imap` extension is mapped to the handler `imap-file` (from [mod_imagemap](#)) and the `.html` extension is mapped to the MIME-type `text/html`, then the file `world.imap.html` will be associated with both the `imap-file` handler and `text/html` MIME-type. When it is processed, the `imap-file` handler will be used, and so it will be treated as a [mod_imagemap](#) imagemap file.

If you would prefer only the last dot-separated part of the filename to be mapped to a particular piece of meta-data, then do not use the `Add*` directives. For example, if you wish to have the file `foo.html.cgi` processed as a CGI script, but not the file `bar.cgi.html`, then instead of using `AddHandler cgi-script .cgi`, use

Configure handler based on final extension only

```
<FilesMatch \.cgi$>  
    SetHandler cgi-script  
</FilesMatch>
```



Content-Encoding

A file of a particular [MIME-type](#) can additionally be encoded a particular way to simplify transmission over the Internet. While this usually will refer to compression, such as `gzip`, it can also refer to encryption, such as `pgp` or to an encoding such as `UUencoding`, which is designed for transmitting a binary file in an ASCII (text) format.

The [HTTP/1.1 RFC](#), section 14.11 puts it this way:

The Content-Encoding entity-header field is used as a modifier to the media-type. When present, its value indicates what additional content codings have been applied to the entity-body, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

By using more than one file extension (see [section above about multiple file extensions](#)), you can indicate that a file is of a particular *type*, and also has a particular *encoding*.

For example, you may have a file which is a Microsoft Word document, which is `pkzipped` to reduce its size. If the `.doc` extension is associated with the Microsoft Word file type, and the `.zip` extension is associated with the `pkzip` file encoding, then the file `Resume.doc.zip` would be known to be a `pkzip`'ed Word document.

Apache sends a `Content-encoding` header with the resource, in order to tell the client browser about the encoding method.

```
Content-encoding: pkzip
```



Character Set and Language

In addition to file type and the file encoding, another important piece of information is what language a particular document is in, and in what character set the file should be displayed. For example, the document might be written in the Vietnamese alphabet, or in Cyrillic, and should be displayed as such. This information, also, is transmitted in HTTP headers.

The character set, language, encoding and mime type are all used in the process of content negotiation (See [mod_negotiation](#)) to determine which document to give to the client, when there are alternative documents in more than one character set, language, encoding or mime type. All filename extensions associations created with [AddCharset](#), [AddEncoding](#), [AddLanguage](#) and [AddType](#) directives (and extensions listed in the [MimeMagicFile](#)) participate in this select process. Filename extensions that are only associated using the [AddHandler](#), [AddInputFilter](#) or [AddOutputFilter](#) directives may be included or excluded from matching by using the [MultiviewsMatch](#) directive.

Charset

To convey this further information, Apache optionally sends a Content - Language header, to specify the language that the document is in, and can append additional information onto the Content - Type header to indicate the particular character set that should be used to correctly render the information.

```
Content-Language: en, fr
Content-Type: text/plain; charset=ISO-8859-1
```

The language specification is the two-letter abbreviation for the language. The char set is the name of the particular character set which should be used.



Description:	Maps the given filename extensions to the specified content charset
Syntax:	<code>AddCharset charset extension [extension]</code> ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The `AddCharset` directive maps the given filename extensions to the specified content charset. *charset* is the [MIME charset parameter](#) of filenames containing *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*.

Example

```
AddLanguage ja .ja
AddCharset EUC-JP .euc
AddCharset ISO-2022-JP .jis
AddCharset SHIFT_JIS .sjis
```

Then the document `xxxx.ja.jis` will be treated as being a Japanese document whose charset is ISO-2022-JP (as will the document `xxxx.jis.ja`). The `AddCharset` directive is useful for both to inform the client about the character encoding of the document so that the document can be interpreted and displayed appropriately, and for [content negotiation](#), where the server returns one from several documents based on the client's charset preference.

The *extension* argument is case-insensitive and can be specified with or without a leading dot. Filenames may have [multiple extensions](#) and the *extension* argument will be compared against each of them.

See also

- mod_negotiation
- AddDefaultCharset



Description:	Maps the given filename extensions to the specified encoding type
Syntax:	AddEncoding <i>MIME-enc extension</i> [<i>extension</i>] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The **AddEncoding** directive maps the given filename extensions to the specified encoding type. *MIME-enc* is the MIME encoding to use for documents containing the *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*.

Example

```
AddEncoding x-gzip .gz
AddEncoding x-compress .Z
```

This will cause filenames containing the .gz extension to be marked as encoded using the x-gzip encoding, and filenames containing the .Z extension to be marked as encoded with x-compress.

Old clients expect x-gzip and x-compress, however the standard dictates that they're equivalent to gzip and compress respectively. Apache does content encoding comparisons by ignoring any leading x-. When responding with an encoding Apache will use whatever form (*i.e.*, x-foo or foo) the client requested. If the client didn't specifically request a particular form Apache will use the form given by the AddEncoding directive. To make this long story short, you should always use x-gzip and x-compress for these two specific encodings. More recent encodings, such as deflate should be specified without the x-.

The *extension* argument is case-insensitive and can be specified with or without a leading dot. Filenames may have [multiple extensions](#) and the *extension* argument will be compared against each of them.



Description:	Maps the filename extensions to the specified handler
Syntax:	<code>AddHandler <i>handler-name</i> <i>extension</i> [<i>extension</i>] ...</code>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

Files having the name *extension* will be served by the specified [handler-name](#). This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*. For example, to activate CGI scripts with the file extension `.cgi`, you might use:

```
AddHandler cgi-script .cgi
```

Once that has been put into your `httpd.conf` file, any file containing the `.cgi` extension will be treated as a CGI program.

The *extension* argument is case-insensitive and can be specified with or without a leading dot. Filenames may have [multiple extensions](#) and the *extension* argument will be compared against each of them.

See also

- [SetHandler](#)



Description:	Maps filename extensions to the filters that will process client requests
Syntax:	<code>AddInputFilter <i>filter</i> [;<i>filter</i>...] <i>extension</i> [<i>extension</i>] ...</code>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	AddInputFilter is only available in Apache 2.0.26 and later.

`AddInputFilter` maps the filename extension *extension* to the [filters](#) which will process client requests and POST input when they are received by the server. This is in addition to any filters defined elsewhere, including the `SetInputFilter` directive. This mapping is merged over any already in force, overriding any mappings that already exist for the same *extension*.

If more than one *filter* is specified, they must be separated by semicolons in the order in which they should process the content. The *filter* is case-insensitive.

The *extension* argument is case-insensitive and can be specified with or without a leading dot. Filenames may have [multiple extensions](#) and the *extension* argument will be compared against each of them.

See also

- [RemoveInputFilter](#)
- [SetInputFilter](#)



Description:	Maps the given filename extension to the specified content language
Syntax:	AddLanguage <i>MIME-lang extension</i> [<i>extension</i>] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The **AddLanguage** directive maps the given filename extension to the specified content language. *MIME-lang* is the MIME language of filenames containing *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*.

Example

```
AddEncoding x-compress .Z
AddLanguage en .en
AddLanguage fr .fr
```

Then the document `xxxx.en.Z` will be treated as being a compressed English document (as will the document `xxxx.Z.en`). Although the content language is reported to the client, the browser is unlikely to use this information. The **AddLanguage** directive is more useful for [content negotiation](#), where the server returns one from several documents based on the client's language preference.

If multiple language assignments are made for the same extension, the last one encountered is the one that is used. That is, for the case of:

```
AddLanguage en .en
AddLanguage en-gb .en
AddLanguage en-us .en
```

documents with the extension `.en` would be treated as being `en-us`.

The *extension* argument is case-insensitive and can be specified with or without a leading dot. Filenames may have [multiple extensions](#) and the *extension* argument will be compared against each of them.

See also

- [mod_negotiation](#)



Description:	Maps filename extensions to the filters that will process responses from the server
Syntax:	<code>AddOutputFilter <i>filter</i> [<i>;filter...</i>] <i>extension</i> [<i>extension</i>] ...</code>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	AddOutputFilter is only available in Apache 2.0.26 and later.

The `AddOutputFilter` directive maps the filename extension *extension* to the [filters](#) which will process responses from the server before they are sent to the client. This is in addition to any filters defined elsewhere, including [SetOutputFilter](#) and [AddOutputFilterByType](#) directive. This mapping is merged over any already in force, overriding any mappings that already exist for the same *extension*.

For example, the following configuration will process all `.shtml` files for server-side includes and will then compress the output using [mod_deflate](#).

```
AddOutputFilter INCLUDES;DEFLATE shtml
```

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content. The *filter* argument is case-insensitive.

The *extension* argument is case-insensitive and can be specified with or without a leading dot. Filenames may have [multiple extensions](#) and the *extension* argument will be compared against each of them.

See also

- [RemoveOutputFilter](#)
- [SetOutputFilter](#)



Description:	Maps the given filename extensions onto the specified content type
Syntax:	AddType <i>MIME-type extension</i> [<i>extension</i>] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The **AddType** directive maps the given filename extensions onto the specified content type. *MIME-type* is the [MIME type](#) to use for filenames containing *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*. This directive can be used to add mappings not listed in the MIME types file (see the [TypesConfig](#) directive).

Example

```
AddType image/gif .gif
```

It is recommended that new MIME types be added using the **AddType** directive rather than changing the [TypesConfig](#) file.

The *extension* argument is case-insensitive and can be specified with or without a leading dot. Filenames may have [multiple extensions](#) and the *extension* argument will be compared against each of them.

See also

- [DefaultType](#)
- [ForceType](#)



Description:	Sets all files in the given scope to the specified language
Syntax:	<code>DefaultLanguage</code> <i>MIME-lang</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The `DefaultLanguage` directive tells Apache that all files in the directive's scope (e.g., all files covered by the current `<Directory>` container) that don't have an explicit language extension (such as `.fr` or `.de` as configured by `AddLanguage`) should be considered to be in the specified *MIME-lang* language. This allows entire directories to be marked as containing Dutch content, for instance, without having to rename each file. Note that unlike using extensions to specify languages, `DefaultLanguage` can only specify a single language.

If no `DefaultLanguage` directive is in force, and a file does not have any language extensions as configured by `AddLanguage`, then that file will be considered to have no language attribute.

Example

```
DefaultLanguage en
```

See also

- [mod_negotiation](#)



Description:	Tells <code>mod_mime</code> to treat <code>path_info</code> components as part of the filename
Syntax:	<code>ModMimeUsePathInfo On Off</code>
Default:	<code>ModMimeUsePathInfo Off</code>
Context:	directory
Status:	Base
Module:	<code>mod_mime</code>
Compatibility:	Available in Apache 2.0.41 and later

The `ModMimeUsePathInfo` directive is used to combine the filename with the `path_info` URL component to apply `mod_mime`'s directives to the request. The default value is `Off` - therefore, the `path_info` component is ignored.

This directive is recommended when you have a virtual filesystem.

Example

```
ModMimeUsePathInfo On
```

If you have a request for `/bar/foo.shtml` where `/bar` is a Location and `ModMimeUsePathInfo` is `On`, `mod_mime` will treat the incoming request as `/bar/foo.shtml` and directives like `AddOutputFilter INCLUDES ..shtml` will add the `INCLUDES` filter to the request. If `ModMimeUsePathInfo` is not set, the `INCLUDES` filter will not be added.

See also

- [AcceptPathInfo](#)



Description:	The types of files that will be included when searching for a matching file with MultiViews
Syntax:	MultiViewsMatch Any NegotiatedOnly Filters Handlers [Handlers Filters]
Default:	MultiViewsMatch NegotiatedOnly
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	Available in Apache 2.0.26 and later.

MultiViewsMatch permits three different behaviors for [mod_negotiation](#)'s MultiViews feature. MultiViews allows a request for a file, e.g. `index.html`, to match any negotiated extensions following the base request, e.g. `index.html.en`, `index.html.fr`, or `index.html.gz`.

The `NegotiatedOnly` option provides that every extension following the base name must correlate to a recognized [mod_mime](#) extension for content negotiation, e.g. `Charset`, `Content-Type`, `Language`, or `Encoding`. This is the strictest implementation with the fewest unexpected side effects, and is the default behavior.

To include extensions associated with `Handlers` and/or `Filters`, set the **MultiViewsMatch** directive to either `Handlers`, `Filters`, or both option keywords. If all other factors are equal, the smallest file will be served, e.g. in deciding between `index.html.cgi` of 500 bytes and `index.html.pl` of 1000 bytes, the `.cgi` file would win in this example. Users of `.asis` files might prefer to use the `Handler` option, if `.asis` files are associated with the `asis`-handler.

You may finally allow Any extensions to match, even if [mod_mime](#) doesn't recognize the extension. This was the behavior in Apache 1.3, and can cause unpredictable results, such as serving .old or .bak files the webmaster never expected to be served.

For example, the following configuration will allow handlers and filters to participate in Multiviews, but will exclude unknown files:

```
MultiviewsMatch Handlers Filters
```

`MultiviewsMatch` is not allowed in a `<Location>` or `<LocationMatch>` section.

See also

- [Options](#)
- [mod_negotiation](#)



Description:	Removes any character set associations for a set of file extensions
Syntax:	<code>RemoveCharset <i>extension</i> [<i>extension</i>] ...</code>
Context:	virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	RemoveCharset is only available in Apache 2.0.24 and later.

The **RemoveCharset** directive removes any character set associations for files with the given extensions. This allows .htaccess files in subdirectories to undo any associations inherited from parent directories or the server config files.

The *extension* argument is case-insensitive and can be specified with or without a leading dot.

Example

```
RemoveCharset .html .shtml
```



Description:	Removes any content encoding associations for a set of file extensions
Syntax:	RemoveEncoding <i>extension</i> [<i>extension</i>] ...
Context:	virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The `RemoveEncoding` directive removes any encoding associations for files with the given extensions. This allows `.htaccess` files in subdirectories to undo any associations inherited from parent directories or the server config files. An example of its use might be:

/foo/.htaccess:

```
AddEncoding x-gzip .gz
AddType text/plain .asc
<Files *.gz.asc>
    RemoveEncoding .gz
</Files>
```

This will cause `foo.gz` to be marked as being encoded with the `gzip` method, but `foo.gz.asc` as an unencoded plaintext file.

Note

`RemoveEncoding` directives are processed *after* any `AddEncoding` directives, so it is possible they may undo the effects of the latter if both occur within the same directory configuration.

The *extension* argument is case-insensitive and can be specified with or without a leading dot.



Description:	Removes any handler associations for a set of file extensions
Syntax:	<code>RemoveHandler <i>extension</i> [<i>extension</i>] ...</code>
Context:	virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The `RemoveHandler` directive removes any handler associations for files with the given extensions. This allows .htaccess files in subdirectories to undo any associations inherited from parent directories or the server config files. An example of its use might be:

/foo/.htaccess:

```
AddHandler server-parsed .html
```

/foo/bar/.htaccess:

```
RemoveHandler .html
```

This has the effect of returning .html files in the /foo/bar directory to being treated as normal files, rather than as candidates for parsing (see the [mod_include](#) module).

The *extension* argument is case-insensitive and can be specified with or without a leading dot.



Description:	Removes any input filter associations for a set of file extensions
Syntax:	<code>RemoveInputFilter <i>extension</i> [<i>extension</i>] ...</code>
Context:	virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	RemoveInputFilter is only available in Apache 2.0.26 and later.

The `RemoveInputFilter` directive removes any input [filter](#) associations for files with the given extensions. This allows .htaccess files in subdirectories to undo any associations inherited from parent directories or the server config files.

The *extension* argument is case-insensitive and can be specified with or without a leading dot.

See also

- [AddInputFilter](#)
- [SetInputFilter](#)



RemoveLanguage Directive

Description:	Removes any language associations for a set of file extensions
Syntax:	RemoveLanguage <i>extension</i> [<i>extension</i>] ...
Context:	virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	RemoveLanguage is only available in Apache 2.0.24 and later.

The **RemoveLanguage** directive removes any language associations for files with the given extensions. This allows .htaccess files in subdirectories to undo any associations inherited from parent directories or the server config files.

The *extension* argument is case-insensitive and can be specified with or without a leading dot.



Description:	Removes any output filter associations for a set of file extensions
Syntax:	<code>RemoveOutputFilter <i>extension</i> [<i>extension</i>]</code> ...
Context:	virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	RemoveOutputFilter is only available in Apache 2.0.26 and later.

The `RemoveOutputFilter` directive removes any output [filter](#) associations for files with the given extensions. This allows `.htaccess` files in subdirectories to undo any associations inherited from parent directories or the server config files.

The *extension* argument is case-insensitive and can be specified with or without a leading dot.

Example

```
RemoveOutputFilter shtml
```

See also

- [AddOutputFilter](#)



Description:	Removes any content type associations for a set of file extensions
Syntax:	<code>RemoveType <i>extension</i> [<i>extension</i>] ...</code>
Context:	virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime

The `RemoveType` directive removes any [MIME type](#) associations for files with the given extensions. This allows `.htaccess` files in subdirectories to undo any associations inherited from parent directories or the server config files. An example of its use might be:

/foo/.htaccess:

```
RemoveType .cgi
```

This will remove any special handling of `.cgi` files in the `/foo/` directory and any beneath it, causing the files to be treated as being of the [DefaultType](#).

Note

`RemoveType` directives are processed *after* any [AddType](#) directives, so it is possible they may undo the effects of the latter if both occur within the same directory configuration.

The *extension* argument is case-insensitive and can be specified with or without a leading dot.



Description:	The location of the <code>mime.types</code> file
Syntax:	<code>TypesConfig file-path</code>
Default:	<code>TypesConfig conf/mime.types</code>
Context:	server config
Status:	Base
Module:	<code>mod_mime</code>

The `TypesConfig` directive sets the location of the [MIME types](#) configuration file. *File-path* is relative to the [ServerRoot](#). This file sets the default list of mappings from filename extensions to content types. Most administrators use the provided `mime.types` file, which associates common filename extensions with IANA registered content types. The current list is maintained at <http://www.iana.org/assignments/media-types/index.html>. This simplifies the `httpd.conf` file by providing the majority of media-type definitions, and may be overridden by `AddType` directives as needed. You should not edit the `mime.types` file, because it may be replaced when you upgrade your server.

The file contains lines in the format of the arguments to an `AddType` directive:

```
MIME-type [extension] ...
```

The case of the extension does not matter. Blank lines, and lines beginning with a hash character (`#`) are ignored.

Please do **not** send requests to the Apache HTTP Server Project to add any new entries in the distributed `mime.types` file unless (1) they are already registered with IANA, and (2) they use widely accepted, non-conflicting filename extensions across platforms. `category/x-subtype` requests will be automatically rejected, as will any new two-letter extensions as they will likely conflict later with the already

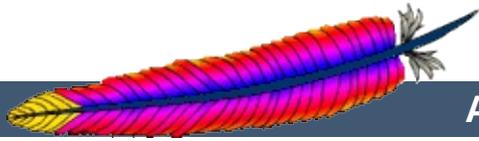
crowded language and character set namespace.

See also

- [mod_mime_magic](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_mime_magic`

Description:	Determines the MIME type of a file by looking at a few bytes of its contents
Status:	Extension
Module Identifier:	<code>mime_magic_module</code>
Source File:	<code>mod_mime_magic.c</code>

Summary

This module determines the [MIME type](#) of files in the same way the Unix `file(1)` command works: it looks at the first few bytes of the file. It is intended as a "second line of defense" for cases that [mod_mime](#) can't resolve.

This module is derived from a free version of the `file(1)` command for Unix, which uses "magic numbers" and other hints from a file's contents to figure out what the contents are. This module is active only if the magic file is specified by the [MimeMagicFile](#) directive.



The contents of the file are plain ASCII text in 4-5 columns. Blank lines are allowed but ignored. Commented lines use a hash mark (#). The remaining lines are parsed for the following columns:

Column	Description																						
1	byte number to begin checking from ">" indicates a dependency upon the previous non-">" line																						
2	type of data to match <table border="1"> <tbody> <tr> <td>byte</td> <td>single character</td> </tr> <tr> <td>short</td> <td>machine-order 16-bit integer</td> </tr> <tr> <td>long</td> <td>machine-order 32-bit integer</td> </tr> <tr> <td>string</td> <td>arbitrary-length string</td> </tr> <tr> <td>date</td> <td>long integer date (seconds since Unix epoch/1970)</td> </tr> <tr> <td>beshort</td> <td>big-endian 16-bit integer</td> </tr> <tr> <td>belong</td> <td>big-endian 32-bit integer</td> </tr> <tr> <td>bedate</td> <td>big-endian 32-bit integer date</td> </tr> <tr> <td>leshort</td> <td>little-endian 16-bit integer</td> </tr> <tr> <td>lelong</td> <td>little-endian 32-bit integer</td> </tr> <tr> <td>ledate</td> <td>little-endian 32-bit integer date</td> </tr> </tbody> </table>	byte	single character	short	machine-order 16-bit integer	long	machine-order 32-bit integer	string	arbitrary-length string	date	long integer date (seconds since Unix epoch/1970)	beshort	big-endian 16-bit integer	belong	big-endian 32-bit integer	bedate	big-endian 32-bit integer date	leshort	little-endian 16-bit integer	lelong	little-endian 32-bit integer	ledate	little-endian 32-bit integer date
byte	single character																						
short	machine-order 16-bit integer																						
long	machine-order 32-bit integer																						
string	arbitrary-length string																						
date	long integer date (seconds since Unix epoch/1970)																						
beshort	big-endian 16-bit integer																						
belong	big-endian 32-bit integer																						
bedate	big-endian 32-bit integer date																						
leshort	little-endian 16-bit integer																						
lelong	little-endian 32-bit integer																						
ledate	little-endian 32-bit integer date																						
3	contents of data to match																						
4	MIME type if matched																						
5	MIME encoding if matched (optional)																						

For example, the following magic file lines would recognize some audio formats:

```
# Sun/NeXT audio data
0      string      .snd
>12    belong      1      audio/basic
>12    belong      2      audio/basic
```

>12	belong	3	audio/basic
>12	belong	4	audio/basic
>12	belong	5	audio/basic
>12	belong	6	audio/basic
>12	belong	7	audio/basic
>12	belong	23	audio/x-adpcm

Or these would recognize the difference between * .doc files containing Microsoft Word or FrameMaker documents. (These are incompatible file formats which use the same file suffix.)

```
# Frame
0 string \<MakerFile          application/x-frame
0 string \<MIFFfile           application/x-frame
0 string \<MakerDictionary    application/x-frame
0 string \<MakerScreenFon     application/x-frame
0 string \<MML                application/x-frame
0 string \<Book               application/x-frame
0 string \<Maker              application/x-frame

# MS-Word
0 string \376\067\0\043       application/msword
0 string \320\317\021\340\241\261 application/msword
0 string \333\245-\0\0\0      application/msword
```

An optional MIME encoding can be included as a fifth column. For example, this can recognize gzipped files and set the encoding for them.

```
# gzip (GNU zip, not to be confused with
#      [Info-ZIP/PKWARE] zip archiver)

0 string \037\213 application/octet-stream x-gzip
```



This module is not for every system. If your system is barely keeping up with its load or if you're performing a web server benchmark, you may not want to enable this because the processing is not free.

However, an effort was made to improve the performance of the original `file(1)` code to make it fit in a busy web server. It was designed for a server where there are thousands of users who publish their own documents. This is probably very common on intranets. Many times, it's helpful if the server can make more intelligent decisions about a file's contents than the file name allows ...even if just to reduce the "why doesn't my page work" calls when users improperly name their own files. You have to decide if the extra work suits your environment.



The following notes apply to the `mod_mime_magic` module and are included here for compliance with contributors' copyright restrictions that require their acknowledgment.

`mod_mime_magic`: MIME type lookup via file magic numbers
Copyright (c) 1996-1997 Cisco Systems, Inc.

This software was submitted by Cisco Systems to the Apache Group in July 1997. Future revisions and derivatives of this source code must acknowledge Cisco Systems as the original contributor of this module. All other licensing and usage conditions are those of the Apache Group.

Some of this code is derived from the free version of the file command originally posted to comp.sources.unix. Copyright info for that program is included below as required.

- Copyright (c) Ian F. Darwin, 1987. Written by Ian F. Darwin.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

1. The author is not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users

ever read sources, credits must appear in the documentation.

4. This notice may not be removed or altered.

For compliance with Mr Darwin's terms: this has been very significantly modified from the free "file" command.

- all-in-one file for compilation convenience when moving from one version of Apache to the next.
- Memory allocation is done through the Apache API's pool structure.
- All functions have had necessary Apache API request or server structures passed to them where necessary to call other Apache API routines. (*i.e.*, usually for logging, files, or memory allocation in itself or a called function.)
- struct magic has been converted from an array to a single-ended linked list because it only grows one record at a time, it's only accessed sequentially, and the Apache API has no equivalent of `realloc()`.
- Functions have been changed to get their parameters from the server configuration instead of globals. (It should be reentrant now but has not been tested in a threaded environment.)
- Places where it used to print results to stdout now saves them in a list where they're used to set the MIME type in the Apache request record.
- Command-line flags have been removed since they will never be used here.



Description:	Enable MIME-type determination based on file contents using the specified magic file
Syntax:	MimeMagicFile <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_mime_magic

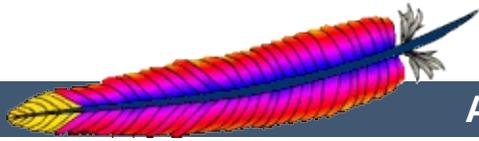
The `MimeMagicFile` directive can be used to enable this module, the default file is distributed at `conf/magic`. Non-rooted paths are relative to the `ServerRoot`. Virtual hosts will use the same file as the main server unless a more specific setting is used, in which case the more specific setting overrides the main server's file.

Example

```
MimeMagicFile conf/magic
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_negotiation

Description:	Provides for content negotiation
Status:	Base
Module Identifier:	negotiation_module
Source File:	mod_negotiation.c

Summary

Content negotiation, or more accurately content selection, is the selection of the document that best matches the clients capabilities, from one of several available documents. There are two implementations of this.

- A type map (a file with the handler type-map) which explicitly lists the files containing the variants.
- A MultiViews search (enabled by the MultiViews [Options](#)), where the server does an implicit filename pattern match, and choose from amongst the results.

See also

[Options](#)

[mod_mime](#)

[Content Negotiation](#)

[Environment Variables](#)



Type Maps

A type map has a format similar to RFC822 mail headers. It contains document descriptions separated by blank lines, with lines beginning with a hash character ('#') treated as comments. A document description consists of several header records; records may be continued on multiple lines if the continuation lines start with spaces. The leading space will be deleted and the lines concatenated. A header record consists of a keyword name, which always ends in a colon, followed by a value. Whitespace is allowed between the header name and value, and between the tokens of value. The headers allowed are:

Content - Encoding:

The encoding of the file. Apache only recognizes encodings that are defined by an [AddEncoding](#) directive. This normally includes the encodings x-compress for compress'd files, and x-gzip for gzip'd files. The x- prefix is ignored for encoding comparisons.

Content - Language:

The language(s) of the variant, as an Internet standard language tag ([RFC 1766](#)). An example is en, meaning English. If the variant contains more than one language, they are separated by a comma.

Content - Length:

The length of the file, in bytes. If this header is not present, then the actual length of the file is used.

Content - Type:

The [MIME media type](#) of the document, with optional parameters. Parameters are separated from the media type and from one another by a semi-colon, with a syntax of name=value. Common parameters include:

level

an integer specifying the version of the media type. For text/html this defaults to 2, otherwise 0.

qs

a floating-point number with a value in the range 0[.000] to 1[.000], indicating the relative 'quality' of this variant compared to the other available variants, independent of the client's capabilities. For example, a jpeg file is usually of higher source quality than an ascii file if it is attempting to represent a photograph. However, if the resource being represented is ascii art, then an ascii file would have a higher source quality than a jpeg file. All qs values are therefore specific to a given resource.

Example

```
Content-Type: image/jpeg; qs=0.8
```

URI :

uri of the file containing the variant (of the given media type, encoded with the given content encoding). These are interpreted as URLs relative to the map file; they must be on the same server (!), and they must refer to files to which the client would be granted access if they were to be requested directly.

Body :

New in Apache 2.0, the actual content of the resource may be included in the type-map file using the Body header. This header must contain a string that designates a delimiter for the body content. Then all following lines in the type map file will be considered part of the resource body until the delimiter string is found.

Example:

```
Body:----xyz----  
<html>  
<body>  
<p>Content of the page.</p>  
</body>  
</html>  
----xyz----
```



A MultiViews search is enabled by the MultiViews [Options](#). If the server receives a request for `/some/dir/foo` and `/some/dir/foo` does *not* exist, then the server reads the directory looking for all files named `foo.*`, and effectively fakes up a type map which names all those files, assigning them the same media types and content-encodings it would have if the client had asked for one of them by name. It then chooses the best match to the client's requirements, and returns that document.

The [MultiViewsMatch](#) directive configures whether Apache will consider files that do not have content negotiation meta-information assigned to them when choosing files.



Description:	Allows content-negotiated documents to be cached by proxy servers
Syntax:	CacheNegotiatedDocs On Off
Default:	CacheNegotiatedDocs Off
Context:	server config, virtual host
Status:	Base
Module:	mod_negotiation
Compatibility:	The syntax changed in version 2.0.

If set, this directive allows content-negotiated documents to be cached by proxy servers. This could mean that clients behind those proxys could retrieve versions of the documents that are not the best match for their abilities, but it will make caching more efficient.

This directive only applies to requests which come from HTTP/1.0 browsers. HTTP/1.1 provides much better control over the caching of negotiated documents, and this directive has no effect in responses to HTTP/1.1 requests.

Prior to version 2.0, `CacheNegotiatedDocs` did not take an argument; it was turned on by the presence of the directive by itself.



ForceLanguagePriority Directive

Description:	Action to take if a single acceptable document is not found
Syntax:	ForceLanguagePriority None Prefer Fallback [Prefer Fallback]
Default:	ForceLanguagePriority Prefer
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_negotiation
Compatibility:	Available in version 2.0.30 and later

The `ForceLanguagePriority` directive uses the given `LanguagePriority` to satisfy negotiation where the server could otherwise not return a single matching document.

`ForceLanguagePriority Prefer` uses `LanguagePriority` to serve a one valid result, rather than returning an HTTP result 300 (MULTIPLE CHOICES) when there are several equally valid choices. If the directives below were given, and the user's `Accept-Language` header assigned `en` and `de` each as quality `.500` (equally acceptable) then the first matching variant, `en`, will be served.

```
LanguagePriority en fr de
ForceLanguagePriority Prefer
```

`ForceLanguagePriority Fallback` uses `LanguagePriority` to serve a valid result, rather than returning an HTTP result 406 (NOT ACCEPTABLE). If the directives below were given, and the user's `Accept-Language` only permitted an `es` language response, but such a variant isn't found, then the first variant from the `LanguagePriority` list below will be served.

LanguagePriority en fr de
ForceLanguagePriority Fallback

Both options, `Prefer` and `Fallback`, may be specified, so either the first matching variant from [LanguagePriority](#) will be served if more than one variant is acceptable, or first available document will be served if none of the variants matched the client's acceptable list of languages.

See also

- [AddLanguage](#)



Description:	The precedence of language variants for cases where the client does not express a preference
Syntax:	LanguagePriority <i>MIME-lang</i> [<i>MIME-lang</i>] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_negotiation

The `LanguagePriority` sets the precedence of language variants for the case where the client does not express a preference, when handling a MultiViews request. The list of *MIME-lang* are in order of decreasing preference.

Example:

```
LanguagePriority en fr de
```

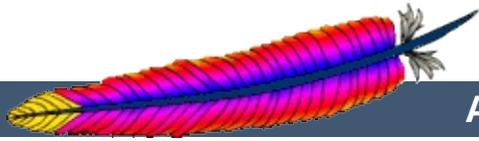
For a request for `foo.html`, where `foo.html.fr` and `foo.html.de` both existed, but the browser did not express a language preference, then `foo.html.fr` would be returned.

Note that this directive only has an effect if a 'best' language cannot be determined by any other means or the `ForceLanguagePriority` directive is not `None`. In general, the client determines the language preference, not the server.

See also

- [AddLanguage](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_nw_ssl

Description:	Enable SSL encryption for NetWare
Status:	Base
Module Identifier:	nwssl_module
Source File:	mod_nw_ssl.c
Compatibility:	NetWare only

Summary

This module enables SSL encryption for a specified port. It takes advantage of the SSL encryption functionality that is built into the NetWare operating system.



Description:	List of additional client certificates
Syntax:	NWSSLTrustedCerts <i>filename</i> [<i>filename</i>] ...
Context:	server config
Status:	Base
Module:	mod_nw_ssl

Specifies a list of client certificate files (DER format) that are used when creating a proxied SSL connection. Each client certificate used by a server must be listed separately in its own .der file.



Description:	Allows a connection to be upgraded to an SSL connection upon request
Syntax:	NWSSLUpgradeable [<i>IP-address</i> :] <i>portnumber</i>
Context:	server config
Status:	Base
Module:	mod_nw_ssl

Allow a connection that was created on the specified address and/or port to be upgraded to an SSL connection upon request from the client. The address and/or port must have already be defined previously with a [Listen](#) directive.

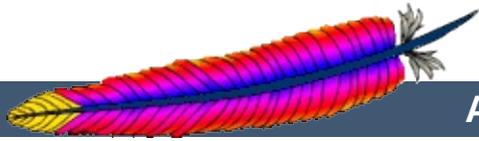


Description:	Enables SSL encryption for the specified port
Syntax:	<code>SecureListen [IP-address:]portnumber Certificate-Name [MUTUAL]</code>
Context:	server config
Status:	Base
Module:	mod_nw_ssl

Specifies the port and the eDirectory based certificate name that will be used to enable SSL encryption. An optional third parameter also enables mutual authentication.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_proxy`

Description:	HTTP/1.1 proxy/gateway server
Status:	Extension
Module Identifier:	<code>proxy_module</code>
Source File:	<code>mod_proxy.c</code>

Summary

Warning

Do not enable proxying with [ProxyRequests](#) until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

This module implements a proxy/gateway for Apache. It implements proxying capability for AJP13 (Apache JServe Protocol version 1.3), FTP, CONNECT (for SSL), HTTP/0.9, HTTP/1.0, and HTTP/1.1. The module can be configured to connect to other proxy modules for these and other protocols.

Apache's proxy features are divided into several modules in addition to `mod_proxy`: `mod_proxy_http`, `mod_proxy_ftp`, `mod_proxy_ajp`, `mod_proxy_balancer`, and `mod_proxy_connect`. Thus, if you want to use one or more of the particular proxy functions, load `mod_proxy` and the appropriate module(s) into the server (either statically at compile-time or dynamically via the [LoadModule](#) directive).

In addition, extended features are provided by other modules. Caching is provided by `mod_cache` and related modules. The ability to contact remote servers using the SSL/TLS protocol is provided by the `SSLProxy*` directives of `mod_ssl`. These additional modules will

need to be loaded and configured to take advantage of these features.

See also

[mod_cache](#)

[mod_proxy_http](#)

[mod_proxy_ftp](#)

[mod_proxy_connect](#)

[mod_proxy_balancer](#)

[mod_ssl](#)



Apache can be configured in both a *forward* and *reverse* proxy (also known as *gateway*) mode.

An ordinary *forward proxy* is an intermediate server that sits between the client and the *origin server*. In order to get content from the origin server, the client sends a request to the proxy naming the origin server as the target. The proxy then requests the content from the origin server and returns it to the client. The client must be specially configured to use the forward proxy to access other sites.

A typical usage of a forward proxy is to provide Internet access to internal clients that are otherwise restricted by a firewall. The forward proxy can also use caching (as provided by [mod_cache](#)) to reduce network usage.

The forward proxy is activated using the [ProxyRequests](#) directive. Because forward proxies allow clients to access arbitrary sites through your server and to hide their true origin, it is essential that you [secure your server](#) so that only authorized clients can access the proxy before activating a forward proxy.

A *reverse proxy* (or *gateway*), by contrast, appears to the client just like an ordinary web server. No special configuration on the client is necessary. The client makes ordinary requests for content in the namespace of the reverse proxy. The reverse proxy then decides where to send those requests and returns the content as if it were itself the origin.

A typical usage of a reverse proxy is to provide Internet users access to a server that is behind a firewall. Reverse proxies can also be used to balance load among several back-end servers or to provide caching for a slower back-end server. In addition, reverse proxies can be used simply to bring several servers into the same URL space.

A reverse proxy is activated using the [ProxyPass](#) directive or the [P]

flag to the RewriteRule directive. It is **not** necessary to turn ProxyRequests on in order to configure a reverse proxy.



The examples below are only a very basic idea to help you get started. Please read the documentation on the individual directives.

In addition, if you wish to have caching enabled, consult the documentation from [mod_cache](#).

Reverse Proxy

```
ProxyPass /foo http://foo.example.com/bar
ProxyPassReverse /foo http://foo.example.com/bar
```

Forward Proxy

```
ProxyRequests On
ProxyVia On
```

```
<Proxy *>
  Order deny,allow
  Deny from all
  Allow from internal.example.com
</Proxy>
```



The proxy manages the configuration of origin servers and their communication parameters in objects called *workers*. There are two built-in workers: the default forward proxy worker and the default reverse proxy worker. Additional workers can be configured explicitly.

The two default workers have a fixed configuration and will be used if no other worker matches the request. They do not use HTTP Keep-Alive or connection pooling. The TCP connections to the origin server will instead be opened and closed for each request.

Explicitly configured workers are identified by their URL. They are usually created and configured using [ProxyPass](#) or [ProxyPassMatch](#) when used for a reverse proxy:

```
ProxyPass /example http://backend.example.com connectiontimeout=5
timeout=30
```

This will create a worker associated with the origin server URL `http://backend.example.com` that will use the given timeout values. When used in a forward proxy, workers are usually defined via the [ProxySet](#) directive:

```
ProxySet http://backend.example.com connectiontimeout=5 timeout=30
```

or alternatively using [Proxy](#) and [ProxySet](#):

```
<Proxy http://backend.example.com>
  ProxySet connectiontimeout=5 timeout=30
</Proxy>
```

Using explicitly configured workers in the forward mode is not very common, because forward proxies usually communicate with many different origin servers. Creating explicit workers for some of the origin servers can still be useful if they are used very often. Explicitly configured

workers have no concept of forward or reverse proxying by themselves. They encapsulate a common concept of communication with origin servers. A worker created by [ProxyPass](#) for use in a reverse proxy will also be used for forward proxy requests whenever the URL to the origin server matches the worker URL, and vice versa.

The URL identifying a direct worker is the URL of its origin server including any path components given:

```
ProxyPass /examples http://backend.example.com/examples
ProxyPass /docs http://backend.example.com/docs
```

This example defines two different workers, each using a separate connection pool and configuration.

Worker Sharing

Worker sharing happens if the worker URLs overlap, which occurs when the URL of some worker is a leading substring of the URL of another worker defined later in the configuration file. In the following example

```
ProxyPass /apps http://backend.example.com/ timeout=60
ProxyPass /examples http://backend.example.com/examples
timeout=10
```

the second worker isn't actually created. Instead the first worker is used. The benefit is, that there is only one connection pool, so connections are more often reused. Note that all configuration attributes given explicitly for the later worker and some configuration defaults will overwrite the configuration given for the first worker. This will be logged as a warning. In the above example, the resulting timeout value for the URL `/apps` will be 10 instead of 60!

If you want to avoid worker sharing, sort your worker definitions by URL length, starting with the longest worker URLs. If you want to

maximize worker sharing, use the reverse sort order. See also the related warning about ordering [ProxyPass](#) directives.

Explicitly configured workers come in two flavors: *direct workers* and *(load) balancer workers*. They support many important configuration attributes which are described below in the [ProxyPass](#) directive. The same attributes can also be set using [ProxySet](#).

The set of options available for a direct worker depends on the protocol which is specified in the origin server URL. Available protocols include `ajp`, `ftp`, `http` and `scgi`.

Balancer workers are virtual workers that use direct workers known as their members to actually handle the requests. Each balancer can have multiple members. When it handles a request, it chooses a member based on the configured load balancing algorithm.

A balancer worker is created if its worker URL uses `balancer` as the protocol scheme. The balancer URL uniquely identifies the balancer worker. Members are added to a balancer using [BalancerMember](#).



Controlling Access to Your Proxy

You can control who can access your proxy via the `<Proxy>` control block as in the following example:

```
<Proxy *>
  Order Deny,Allow
  Deny from all
  Allow from 192.168.0
</Proxy>
```

For more information on access control directives, see [mod_authz_host](#).

Strictly limiting access is essential if you are using a forward proxy (using the `ProxyRequests` directive). Otherwise, your server can be used by any client to access arbitrary hosts while hiding his or her true identity. This is dangerous both for your network and for the Internet at large. When using a reverse proxy (using the `ProxyPass` directive with `ProxyRequests Off`), access control is less critical because clients can only contact the hosts that you have specifically configured.



If you're using the [ProxyBlock](#) directive, hostnames' IP addresses are looked up and cached during startup for later match test. This may take a few seconds (or more) depending on the speed with which the hostname lookups occur.



Intranet Proxy

An Apache proxy server situated in an intranet needs to forward external requests through the company's firewall (for this, configure the [ProxyRemote](#) directive to forward the respective *scheme* to the firewall proxy). However, when it has to access resources within the intranet, it can bypass the firewall when accessing hosts. The [NoProxy](#) directive is useful for specifying which hosts belong to the intranet and should be accessed directly.

Users within an intranet tend to omit the local domain name from their WWW requests, thus requesting "http://somehost/" instead of `http://somehost.example.com/`. Some commercial proxy servers let them get away with this and simply serve the request, implying a configured local domain. When the [ProxyDomain](#) directive is used and the server is [configured for proxy service](#), Apache can return a redirect response and send the client to the correct, fully qualified, server address. This is the preferred method since the user's bookmark files will then contain fully qualified hosts.



Prerequisites

For circumstances where `mod_proxy` is sending requests to an origin server that doesn't properly implement keepalives or HTTP/1.1, there are two [environment variables](#) that can force the request to use HTTP/1.0 with no keepalive. These are set via the `SetEnv` directive.

These are the `force-proxy-request-1.0` and `proxy-nokeepalive` notes.

```
<Location /buggyappserver/>
  ProxyPass http://buggyappserver:7001/foo/
  SetEnv force-proxy-request-1.0 1
  SetEnv proxy-nokeepalive 1
</Location>
```



Request Bodies

Some request methods such as POST include a request body. The HTTP protocol requires that requests which include a body either use chunked transfer encoding or send a Content - Length request header. When passing these requests on to the origin server, [mod_proxy_http](#) will always attempt to send the Content - Length. But if the body is large and the original request used chunked encoding, then chunked encoding may also be used in the upstream request. You can control this selection using [environment variables](#). Setting proxy-sendcl ensures maximum compatibility with upstream servers by always sending the Content - Length, while setting proxy-sendchunked minimizes resource usage by using chunked encoding.



When acting in a reverse-proxy mode (using the [ProxyPass](#) directive, for example), [mod_proxy_http](#) adds several request headers in order to pass information to the origin server. These headers are:

X-Forwarded-For

The IP address of the client.

X-Forwarded-Host

The original host requested by the client in the Host HTTP request header.

X-Forwarded-Server

The hostname of the proxy server.

Be careful when using these headers on the origin server, since they will contain more than one (comma-separated) value if the original request already contained one of these headers. For example, you can use `%{X-Forwarded-For}i` in the log format string of the origin server to log the original clients IP address, but you may get more than one address if the request passes through several proxies.

See also the [ProxyPreserveHost](#) and [ProxyVia](#) directives, which control other request headers.



Description:	Ports that are allowed to CONNECT through the proxy
Syntax:	<code>AllowCONNECT port [port] ...</code>
Default:	<code>AllowCONNECT 443 563</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_proxy</code>

The `AllowCONNECT` directive specifies a list of port numbers to which the proxy CONNECT method may connect. Today's browsers use this method when a `https` connection is requested and proxy tunneling over HTTP is in effect.

By default, only the default `https` port (443) and the default `snews` port (563) are enabled. Use the `AllowCONNECT` directive to override this default and allow connections to the listed ports only.

Note that you'll need to have `mod_proxy_connect` present in the server in order to get the support for the CONNECT at all.



Description:	Add a member to a load balancing group
Syntax:	<code>BalancerMember [<i>balancerurl</i>] <i>url</i> [<i>key=value</i> [<i>key=value</i> ...]]</code>
Context:	directory
Status:	Extension
Module:	mod_proxy
Compatibility:	BalancerMember is only available in Apache 2.2 and later.

This directive adds a member to a load balancing group. It can be used within a `<Proxy balancer://...>` container directive and can take any of the key value pairs available to [ProxyPass](#) directives.

The *balancerurl* is only needed when not within a `<Proxy balancer://...>` container directive. It corresponds to the url of a balancer defined in [ProxyPass](#) directive.



Description:	Hosts, domains, or networks that will be connected to directly
Syntax:	NoProxy <i>host</i> [<i>host</i>] ...
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This directive is only useful for Apache proxy servers within intranets. The **NoProxy** directive specifies a list of subnets, IP addresses, hosts and/or domains, separated by spaces. A request to a host which matches one or more of these is always served directly, without forwarding to the configured **ProxyRemote** proxy server(s).

Example

```
ProxyRemote * http://firewall.example.com:81
NoProxy .example.com 192.168.112.0/21
```

The *host* arguments to the **NoProxy** directive are one of the following type list:

Domain

A *Domain* is a partially qualified DNS domain name, preceded by a period. It represents a list of hosts which logically belong to the same DNS domain or zone (*i.e.*, the suffixes of the hostnames are all ending in *Domain*).

Examples

```
.com .apache.org.
```

To distinguish *Domains* from **Hostnames** (both syntactically and semantically; a DNS domain can have a DNS A record, too!), *Domains* are always written with a leading period.

Note

Domain name comparisons are done without regard to the case, and *Domains* are always assumed to be anchored in the root of the DNS tree; therefore, the two domains `.ExAmPlE.com` and `.example.com.` (note the trailing period) are considered equal. Since a domain comparison does not involve a DNS lookup, it is much more efficient than subnet comparison.

SubNet

A *SubNet* is a partially qualified internet address in numeric (dotted quad) form, optionally followed by a slash and the netmask, specified as the number of significant bits in the *SubNet*. It is used to represent a subnet of hosts which can be reached over a common network interface. In the absence of the explicit net mask it is assumed that omitted (or zero valued) trailing digits specify the mask. (In this case, the netmask can only be multiples of 8 bits wide.) Examples:

192.168 or 192.168.0.0

the subnet 192.168.0.0 with an implied netmask of 16 valid bits (sometimes used in the netmask form 255.255.0.0)

192.168.112.0/21

the subnet 192.168.112.0/21 with a netmask of 21 valid bits (also used in the form 255.255.248.0)

As a degenerate case, a *SubNet* with 32 valid bits is the equivalent to an [IPAddr](#), while a *SubNet* with zero valid bits (e.g., 0.0.0.0/0) is the same as the constant `_Default_`, matching any IP address.

IPAddr

A *IPAddr* represents a fully qualified internet address in numeric (dotted quad) form. Usually, this address represents a host, but there need not necessarily be a DNS domain name connected with the address.

Example

192.168.123.7

Note

An *IPAddr* does not need to be resolved by the DNS system, so it can result in more effective apache performance.

Hostname

A *Hostname* is a fully qualified DNS domain name which can be resolved to one or more [IPAddrs](#) via the DNS domain name service. It represents a logical host (in contrast to [Domains](#), see above) and must be resolvable to at least one [IPAddr](#) (or often to a list of hosts with different [IPAddrs](#)).

Examples

prep.ai.example.com
www.apache.org

Note

In many situations, it is more effective to specify an [IPAddr](#) in place of a *Hostname* since a DNS lookup can be avoided. Name resolution in Apache can take a remarkable deal of time when the connection to the name server uses a slow PPP link.

Hostname comparisons are done without regard to the case, and *Hostnames* are always assumed to be anchored in the root of the DNS tree; therefore, the two hosts `www.ExAmPle.com` and `www.example.com.` (note the trailing period) are considered equal.

See also

- [DNS Issues](#)



Description:	Container for directives applied to proxied resources
Syntax:	<code><Proxy <i>wildcard-url</i>> . . .</Proxy></code>
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

Directives placed in `<Proxy>` sections apply only to matching proxied content. Shell-style wildcards are allowed.

For example, the following will allow only hosts in `yournetwork.example.com` to access content via your proxy server:

```
<Proxy *>
  Order Deny,Allow
  Deny from all
  Allow from yournetwork.example.com
</Proxy>
```

The following example will process all files in the `foo` directory of `example.com` through the `INCLUDES` filter when they are sent through the proxy server:

```
<Proxy http://example.com/foo/*>
  SetOutputFilter INCLUDES
</Proxy>
```

Differences from the Location configuration section

A backend URL matches the configuration section if it begins with the `wildcard-url` string, even if the last path segment in the directive only matches a prefix of the backend URL. For example, `<Proxy http://example.com/foo>` matches all of `http://example.com/foo`, `http://example.com/foo/bar`, and `http://example.com/foobar`. The matching of the final URL differs from the behavior of the `<Location>` section, which for purposes of this note treats the final path component

as if it ended in a slash.

For more control over the matching, see [<ProxyMatch>](#).

See also

- [<ProxyMatch>](#)



ProxyBadHeader Directive

Description:	Determines how to handle bad header lines in a response
Syntax:	ProxyBadHeader IsError Ignore StartBody
Default:	ProxyBadHeader IsError
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in Apache 2.0.44 and later

The **ProxyBadHeader** directive determines the behavior of [mod_proxy](#) if it receives syntactically invalid response header lines (*i.e.* containing no colon) from the origin server. The following arguments are possible:

IsError

Abort the request and end up with a 502 (Bad Gateway) response. This is the default behavior.

Ignore

Treat bad header lines as if they weren't sent.

StartBody

When receiving the first bad header line, finish reading the headers and treat the remainder as body. This helps to work around buggy backend servers which forget to insert an empty line between the headers and the body.



Description: Words, hosts, or domains that are banned from being proxied

Syntax: ProxyBlock * |*word*|*host*|*domain*
[*word*|*host*|*domain*] ...

Context: server config, virtual host

Status: Extension

Module: mod_proxy

The **ProxyBlock** directive specifies a list of words, hosts and/or domains, separated by spaces. HTTP, HTTPS, and FTP document requests to sites whose names contain matched words, hosts or domains are *blocked* by the proxy server. The proxy module will also attempt to determine IP addresses of list items which may be hostnames during startup, and cache them for match test as well. That may slow down the startup time of the server.

Example

```
ProxyBlock joes-garage.com some-host.co.uk rocky.wotsamattau.edu
```

rocky.wotsamattau.edu would also be matched if referenced by IP address.

Note that wotsamattau would also be sufficient to match wotsamattau.edu.

Note also that

```
ProxyBlock *
```

blocks connections to all sites.



Description:	Default domain name for proxied requests
Syntax:	ProxyDomain <i>Domain</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This directive is only useful for Apache proxy servers within intranets. The `ProxyDomain` directive specifies the default domain which the apache proxy server will belong to. If a request to a host without a domain name is encountered, a redirection response to the same host with the configured *Domain* appended will be generated.

Example

```
ProxyRemote * http://firewall.example.com:81
NoProxy .example.com 192.168.112.0/21
ProxyDomain .example.com
```



Description:	Override error pages for proxied content
Syntax:	ProxyErrorOverride On Off
Default:	ProxyErrorOverride Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in version 2.0 and later

This directive is useful for reverse-proxy setups where you want to have a common look and feel on the error pages seen by the end user. This also allows for included files (via `mod_include`'s SSI) to get the error code and act accordingly. (Default behavior would display the error page of the proxied server. Turning this on shows the SSI Error message.)

This directive does not affect the processing of informational (1xx), normal success (2xx), or redirect (3xx) responses.



ProxyFtpDirCharset Directive

Description:	Define the character set for proxied FTP listings
Syntax:	ProxyFtpDirCharset <i>character set</i>
Default:	ProxyFtpDirCharset ISO-8859-1
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in Apache 2.2.7 and later

The `ProxyFtpDirCharset` directive defines the character set to be set for FTP directory listings in HTML generated by `mod_proxy_ftp`.



Description:	Determine size of internal data throughput buffer
Syntax:	ProxyIOBufferSize <i>bytes</i>
Default:	ProxyIOBufferSize 8192
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

The `ProxyIOBufferSize` directive adjusts the size of the internal buffer which is used as a scratchpad for the data between input and output. The size must be at least 8192.

When the `mod_proxy_ajp` module is used, this value is aligned to a 1024 byte boundary, and values larger than 65536 are set to 65536 in accordance with the AJP protocol.

In almost every case, there's no reason to change that value.



ProxyMatch Directive

Description:	Container for directives applied to regular-expression-matched proxied resources
Syntax:	<code><ProxyMatch regex> ...</ProxyMatch></code>
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

The `<ProxyMatch>` directive is identical to the `<Proxy>` directive, except that it matches URLs using [regular expressions](#).

See also

- [<Proxy>](#)



ProxyMaxForwards Directive

Description:	Maximum number of proxies that a request can be forwarded through
Syntax:	<code>ProxyMaxForwards <i>number</i></code>
Default:	<code>ProxyMaxForwards -1</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_proxy</code>
Compatibility:	Available in Apache 2.0 and later; default behaviour changed in 2.2.7

The `ProxyMaxForwards` directive specifies the maximum number of proxies through which a request may pass if there's no `Max-Forwards` header supplied with the request. This may be set to prevent infinite proxy loops or a DoS attack.

Example

```
ProxyMaxForwards 15
```

Note that setting `ProxyMaxForwards` is a violation of the HTTP/1.1 protocol (RFC2616), which forbids a Proxy setting `Max-Forwards` if the Client didn't set it. Earlier Apache versions would always set it. A negative `ProxyMaxForwards` value, including the default `-1`, gives you protocol-compliant behavior but may leave you open to loops.



Description:	Maps remote servers into the local server URL-space
Syntax:	ProxyPass [<i>path</i>] ! <i>url</i> [<i>key=value</i> [<i>key=value ...</i>]] [nocanon] [interpolate]
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_proxy

This directive allows remote servers to be mapped into the space of the local server. The local server does not act as a proxy in the conventional sense but appears to be a mirror of the remote server. The local server is often called a *reverse proxy* or *gateway*. The *path* is the name of a local virtual path; *url* is a partial URL for the remote server and cannot include a query string.

The [ProxyRequests](#) directive should usually be set **off** when using [ProxyPass](#).

Suppose the local server has address `http://example.com/`; then

```
ProxyPass /mirror/foo/ http://backend.example.com/
```

will cause a local request for `http://example.com/mirror/foo/bar` to be internally converted into a proxy request to `http://backend.example.com/bar`.

If the first argument ends with a trailing */*, the second argument should also end with a trailing */*, and vice versa. Otherwise, the resulting requests to the backend may miss some needed slashes and do not deliver the expected results.

When used inside a [<Location>](#) section, the first argument is omitted

and the local directory is obtained from the `<Location>`. The same will occur inside a `<LocationMatch>` section; however, ProxyPass does not interpret the regexp as such, so it is necessary to use `ProxyPassMatch` in this situation instead.

The ProxyPass directive is not supported in `<Directory>` or `<Files>` sections.

If you require a more flexible reverse-proxy configuration, see the `RewriteRule` directive with the [P] flag.

The ! directive is useful in situations where you don't want to reverse-proxy a subdirectory, e.g.

```
ProxyPass /mirror/foo/i !  
ProxyPass /mirror/foo http://backend.example.com
```

will proxy all requests to `/mirror/foo` to `backend.example.com` *except* requests made to `/mirror/foo/i`.

Ordering ProxyPass Directives

The configured `ProxyPass` and `ProxyPassMatch` rules are checked in the order of configuration. The first rule that matches wins. So usually you should sort conflicting `ProxyPass` rules starting with the longest URLs first. Otherwise, later rules for longer URLs will be hidden by any earlier rule which uses a leading substring of the URL. Note that there is some relation with worker sharing.

For the same reasons, exclusions must come *before* the general `ProxyPass` directives.

Ordering ProxyPass and RewriteRule Directives

`RewriteRule` directives are evaluated before `ProxyPass` ones.

ProxyPass key=value Parameters

In Apache HTTP Server 2.1 and later, `mod_proxy` supports pooled connections to a backend server. Connections created on demand can be retained in a pool for future use. Limits on the pool size and other settings can be coded on the `ProxyPass` directive using `key=value` parameters, described in the tables below.

By default, `mod_proxy` will allow and retain the maximum number of connections that could be used simultaneously by that web server child process. Use the `max` parameter to reduce the number from the default. Use the `ttl` parameter to set an optional time to live; connections which have been unused for at least `ttl` seconds will be closed. `ttl` can be used to avoid using a connection which is subject to closing because of the backend server's keep-alive timeout.

The pool of connections is maintained per web server child process, and `max` and other settings are not coordinated among all child processes, except when only one child process is allowed by configuration or MPM design.

Example

```
ProxyPass /example http://backend.example.com max=20 ttl=120 retry=300
```

Parameter	Default	Description
<code>min</code>	0	Minimum number of connection pool entries, unrelated to the actual number of connections. This only needs to be modified from the default for special circumstances where heap memory associated with the backend connections should be preallocated or retained.
<code>max</code>	1...n	Maximum number of connections

smax	max	<p>that will be allowed to the backend server. The default for this limit is the number of threads per process in the active MPM. In the Prefork MPM, this is always 1; while with other MPMs, it is controlled by the <code>ThreadsPerChild</code> directive.</p>
acquire	-	<p>Retained connection pool entries above this limit are freed during certain operations if they have been unused for longer than the time to live, controlled by the <code>ttl</code> parameter. If the connection pool entry has an associated connection, it will be closed. This only needs to be modified from the default for special circumstances where connection pool entries and any associated connections which have exceeded the time to live need to be freed or closed more aggressively.</p>
connectiontimeout	timeout	<p>If set, this will be the maximum time to wait for a free connection in the connection pool, in milliseconds. If there are no free connections in the pool, the Apache will return <code>SERVER_BUSY</code> status to the client.</p> <p>Connect timeout in seconds. The number of seconds Apache waits for the creation of a connection to the backend to complete. By adding a postfix of <code>ms</code>, the timeout</p>

disablereuse	Off	can be also set in milliseconds. This parameter should be used when you want to force mod_proxy to immediately close a connection to the backend after being used, and thus, disable its persistent connection and pool for that backend. This helps in various situations where a firewall between Apache and the backend server (regardless of protocol) tends to silently drop connections or when backends themselves may be under round-robin DNS. To disable connection pooling reuse, set this property value to On.
flushpackets	off	Determines whether the proxy module will auto-flush the output brigade after each "chunk" of data. 'off' means that it will flush only when needed; 'on' means after each chunk is sent; and 'auto' means poll/wait for a period of time and flush if no input has been received for 'flushwait' milliseconds. Currently, this is in effect only for AJP.
flushwait	10	The time to wait for additional input, in milliseconds, before flushing the output brigade if 'flushpackets' is 'auto'.
keepalive	Off	This parameter should be used when you have a firewall between

your Apache and the backend server, which tends to drop inactive connections. This flag will tell the Operating System to send KEEP_ALIVE messages on inactive connections and thus prevent the firewall from dropping the connection. To enable keepalive, set this property value to On.

The frequency of initial and subsequent TCP keepalive probes depends on global OS settings, and may be as high as 2 hours. To be useful, the frequency configured in the OS must be smaller than the threshold used by the firewall.

lbset	0	Sets the load balancer cluster set that the worker is a member of. The load balancer will try all members of a lower numbered lbset before trying higher numbered ones.
ping	0	Ping property tells webserver to send a CPING request on ajp13 connection before forwarding a request. The parameter is the delay in seconds to wait for the CPONG reply. This features has been added to avoid problem with hung and busy Tomcat's and require ajp13 ping/pong support which has been implemented on Tomcat

3.3.2+, 4.1.28+ and 5.0.13+. This will increase the network traffic during the normal operation which could be an issue, but it will lower the traffic in case some of the cluster nodes are down or busy. Currently, this has an effect only for AJP. By adding a postfix of ms, the delay can be also set in milliseconds.

loadfactor 1

Worker load factor. Used with BalancerMember. It is a number between 1 and 100 and defines the normalized weighted load applied to the worker.

redirect -

Redirection Route of the worker. This value is usually set dynamically to enable safe removal of the node from the cluster. If set, all requests without session id will be redirected to the BalancerMember that has route parameter equal to this value.

retry 60

Connection pool worker retry timeout in seconds. If the connection pool worker to the backend server is in the error state, Apache will not forward any requests to that server until the timeout expires. This enables to shut down the backend server for maintenance and bring it back online later. A value of 0 means always retry workers in an error

		state with no timeout.
route	-	Route of the worker when used inside load balancer. The route is a value appended to session id.
status	-	Single letter value defining the initial status of this worker: 'D' is disabled; 'S' is stopped; 'I' is ignore-errors; 'H' is hot-standby; and 'E' is in an error state. Status can be set (which is the default) by prepending with '+' or cleared by prepending with '-'. Thus, a setting of 'S-E' sets this worker to Stopped and clears the in-error flag.
timeout	ProxyTimeout	Connection timeout in seconds. The number of seconds Apache waits for data sent by / to the backend.
ttl	-	Time to live for inactive connections and associated connection pool entries, in seconds. Once reaching this limit, a connection will not be used again; it will be closed at some later time.

If the [ProxyPass](#) directive scheme starts with the balancer :// (eg: balancer://cluster/, any path information is ignored), then a virtual worker that does not really communicate with the backend server will be created. Instead, it is responsible for the management of several "real" workers. In that case, the special set of parameters can be added to this virtual worker. See [mod_proxy_balancer](#) for more information about how the balancer works.

Parameter	Default	Description
lbmethod	byrequests	Balancer load-balance method. Select the load-balancing scheduler method to use. Either <code>byrequests</code> , to perform weighted request counting; <code>bytraffic</code> , to perform weighted traffic byte count balancing; or <code>bybusyness</code> (Apache HTTP Server 2.2.10 and later), to perform pending request balancing. The default is <code>byrequests</code> .
maxattempts	One less than the number of workers, or 1 with a single worker.	Maximum number of failover attempts before giving up.
nofailover	Off	If set to <code>On</code> , the session will break if the worker is in error state or disabled. Set this value to <code>On</code> if backend servers do not support session replication.
stickysession	-	Balancer sticky session name. The value is usually set to something like <code>JSESSIONID</code> or <code>PHPSESSIONID</code> , and it depends on the backend application server that support sessions. If the backend application server uses different names for cookies and url encoded id (like servlet containers), use <code> </code> to separate them. The first part is for the cookie; the second for the path.
scolonpathdelim	Off	If set to <code>On</code> , the semi-colon character <code>;</code> will be used as an additional sticky session

		path delimiter/separator. This is mainly used to emulate mod_jk's behavior when dealing with paths such as JSESSIONID=6736bcf34;foo=aabfa
timeout	0	Balancer timeout in seconds. If set, this will be the maximum time to wait for a free worker. The default is to not wait.
failonstatus	-	A single or comma-separated list of HTTP status codes. If set, this will force the worker into error state when the backend returns any status code in the list. Worker recovery behaves the same as other worker errors. Available with Apache HTTP Server 2.2.17 and later.
failontimeout	Off	If set, an IO read timeout after a request is sent to the backend will force the worker into error state. Worker recovery behaves the same as other worker errors. Available with Apache HTTP Server 2.2.25 and later.
forcerecovery	On	Force the immediate recovery of all workers without considering the retry parameter of the workers if all workers of a balancer are in error state. There might be cases where an already overloaded backend can get into deeper trouble if the recovery of all workers is enforced without considering the retry parameter of each worker. In this case, set to Off. Available with Apache HTTP Server 2.2.23 and later.

A sample balancer setup:

```
ProxyPass /special-area http://special.example.com smax=5 max=10
```

```
ProxyPass / balancer://mycluster/ stickysession=JSESSIONID|jsessionid
nofailover=On
<Proxy balancer://mycluster>
  BalancerMember ajp://1.2.3.4:8009
  BalancerMember ajp://1.2.3.5:8009 loadfactor=20
  # Less powerful server, don't send as many requests there,
  BalancerMember ajp://1.2.3.6:8009 loadfactor=5
</Proxy>
```

Setting up a hot-standby that will only be used if no other members are available:

```
ProxyPass / balancer://hotcluster/
<Proxy balancer://hotcluster>
  BalancerMember ajp://1.2.3.4:8009 loadfactor=1
  BalancerMember ajp://1.2.3.5:8009 loadfactor=2
  # The below is the hot standby
  BalancerMember ajp://1.2.3.6:8009 status=+H
  ProxySet lbmethod=bytraffic
</Proxy>
```

Additional ProxyPass Keywords

Normally, `mod_proxy` will canonicalise ProxyPassed URLs. But this may be incompatible with some backends, particularly those that make use of `PATH_INFO`. The optional `nocanon` keyword suppresses this and passes the URL path "raw" to the backend. Note that this keyword may affect the security of your backend, as it removes the normal limited protection against URL-based attacks provided by the proxy.

The optional `interpolate` keyword (available in `httpd 2.2.9` and later), in combination with `ProxyPassInterpolateEnv`, causes the `ProxyPass` to interpolate environment variables, using the syntax `${VARIABLE}`. Note that many of the standard CGI-derived environment variables will not exist when this interpolation happens, so you may still have to resort to `mod_rewrite` for complex rules. Also note that interpolation is not supported within the scheme portion of a URL. Dynamic determination of the scheme can be accomplished with `mod_rewrite` as in the following example.

RewriteEngine On

RewriteCond %{HTTPS} =off

RewriteRule . - [E=protocol:http]

RewriteCond %{HTTPS} =on

RewriteRule . - [E=protocol:https]

RewriteRule ^/mirror/foo/(.*) %{ENV:protocol}://b

ProxyPassReverse /mirror/foo/ http://backend.exa

ProxyPassReverse /mirror/foo/ https://backend.exa



ProxyPassInterpolateEnv Directive

Description:	Enable Environment Variable interpolation in Reverse Proxy configurations
Syntax:	ProxyPassInterpolateEnv On Off
Default:	ProxyPassInterpolateEnv Off
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in httpd 2.2.9 and later

This directive, together with the *interpolate* argument to `ProxyPass`, `ProxyPassReverse`, `ProxyPassReverseCookieDomain`, and `ProxyPassReverseCookiePath`, enables reverse proxies to be dynamically configured using environment variables which may be set by another module such as `mod_rewrite`. It affects the `ProxyPass`, `ProxyPassReverse`, `ProxyPassReverseCookieDomain`, and `ProxyPassReverseCookiePath` directives and causes them to substitute the value of an environment variable `varname` for the string `${varname}` in configuration directives if the *interpolate* option is set.

Keep this turned off (for server performance) unless you need it!



ProxyPassMatch Directive

Description:	Maps remote servers into the local server URL-space using regular expressions
Syntax:	<code>ProxyPassMatch [regex] ! url [key=value [key=value ...]]</code>
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_proxy
Compatibility:	available in Apache 2.2.5 and later

This directive is equivalent to [ProxyPass](#) but makes use of regular expressions instead of simple prefix matching. The supplied regular expression is matched against the *url*, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a new *url*.

Suppose the local server has address `http://example.com/`; then

```
ProxyPassMatch ^(/.*\.gif)$ http://backend.example.com$1
```

will cause a local request for `http://example.com/foo/bar.gif` to be internally converted into a proxy request to `http://backend.example.com/foo/bar.gif`.

Note

The URL argument must be parsable as a URL *before* regexp substitutions (as well as after). This limits the matches you can use. For instance, if we had used

```
ProxyPassMatch ^(/.*\.gif)$ http://backend.example.com:8000$1
```

in our previous example, it would fail with a syntax error at server

startup. This is a bug (PR 46665 in the ASF bugzilla), and the workaround is to reformulate the match:

```
ProxyPassMatch ^/(.*\.gif)$ http://backend.example.com:8000/$1
```

The `!` directive is useful in situations where you don't want to reverse-proxy a subdirectory.

When used inside a `<LocationMatch>` section, the first argument is omitted and the regexp is obtained from the `<LocationMatch>`.

If you require a more flexible reverse-proxy configuration, see the `RewriteRule` directive with the `[P]` flag.

Security Warning

Take care when constructing the target URL of the rule, considering the security impact from allowing the client influence over the set of URLs to which your server will act as a proxy. Ensure that the scheme and hostname part of the URL is either fixed or does not allow the client undue influence.



Description:	Adjusts the URL in HTTP response headers sent from a reverse proxied server
Syntax:	<code>ProxyPassReverse [path] url [interpolate]</code>
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_proxy

This directive lets Apache adjust the URL in the `Location`, `Content-Location` and `URI` headers on HTTP redirect responses. This is essential when Apache is used as a reverse proxy (or gateway) to avoid bypassing the reverse proxy because of HTTP redirects on the backend servers which stay behind the reverse proxy.

Only the HTTP response headers specifically mentioned above will be rewritten. Apache will not rewrite other response headers, nor will it rewrite URL references inside HTML pages. This means that if the proxied content contains absolute URL references, they will bypass the proxy. A third-party module that will look inside the HTML and rewrite URL references is Nick Kew's [mod_proxy_html](#).

path is the name of a local virtual path; *url* is a partial URL for the remote server. These parameters are used the same way as for the [ProxyPass](#) directive.

For example, suppose the local server has address `http://example.com/`; then

```
ProxyPass /mirror/foo/ http://backend.example.com/
ProxyPassReverse /mirror/foo/ http://backend.example.com/
ProxyPassReverseCookieDomain backend.example.com public.example.com
ProxyPassReverseCookiePath / /mirror/foo/
```

will not only cause a local request for the `http://example.com/mirror/foo/bar` to be internally converted

into a proxy request to `http://backend.example.com/bar` (the functionality which ProxyPass provides here). It also takes care of redirects which the server `backend.example.com` sends when redirecting `http://backend.example.com/bar` to `http://backend.example.com/quux`. Apache adjusts this to `http://example.com/mirror/foo/quux` before forwarding the HTTP redirect response to the client. Note that the hostname used for constructing the URL is chosen in respect to the setting of the [UseCanonicalName](#) directive.

Note that this [ProxyPassReverse](#) directive can also be used in conjunction with the proxy feature (`RewriteRule ... [P]`) from [mod_rewrite](#) because it doesn't depend on a corresponding [ProxyPass](#) directive.

The optional *interpolate* keyword (available in httpd 2.2.9 and later), used together with [ProxyPassInterpolateEnv](#), enables interpolation of environment variables specified using the format `${VARIABLE}`. Note that interpolation is not supported within the scheme portion of a URL.

When used inside a [<Location>](#) section, the first argument is omitted and the local directory is obtained from the [<Location>](#). The same occurs inside a [<LocationMatch>](#) section, but will probably not work as intended, as ProxyPassReverse will interpret the regexp literally as a path; if needed in this situation, specify the ProxyPassReverse outside the section or in a separate [<Location>](#) section.

This directive is not supported in [<Directory>](#) or [<Files>](#) sections.



Description: Adjusts the Domain string in Set-Cookie headers from a reverse- proxied server

Syntax: ProxyPassReverseCookieDomain *internal-domain public-domain [interpolate]*

Context: server config, virtual host, directory

Status: Extension

Module: mod_proxy

Usage is basically similar to [ProxyPassReverse](#), but instead of rewriting headers that are a URL, this rewrites the domain string in Set - Cookie headers.



Description:	Adjusts the Path string in Set-Cookie headers from a reverse- proxied server
Syntax:	ProxyPassReverseCookiePath <i>internal-path</i> <i>public-path</i> [<i>interpolate</i>]
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_proxy

Useful in conjunction with [ProxyPassReverse](#) in situations where backend URL paths are mapped to public paths on the reverse proxy. This directive rewrites the path string in Set -Cookie headers. If the beginning of the cookie path matches *internal-path*, the cookie path will be replaced with *public-path*.

In the example given with [ProxyPassReverse](#), the directive:

```
ProxyPassReverseCookiePath / /mirror/foo/
```

will rewrite a cookie with backend path / (or /example or, in fact, anything) to /mirror/foo/.



Description:	Use incoming Host HTTP request header for proxy request
Syntax:	ProxyPreserveHost On Off
Default:	ProxyPreserveHost Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in Apache 2.0.31 and later.

When enabled, this option will pass the Host: line from the incoming request to the proxied host, instead of the hostname specified in the `ProxyPass module="mod_proxy"` line.

This option should normally be turned Off. It is mostly useful in special configurations like proxied mass name-based virtual hosting, where the original Host header needs to be evaluated by the backend server.



Description: Network buffer size for proxied HTTP and FTP connections

Syntax: ProxyReceiveBufferSize *bytes*

Default: ProxyReceiveBufferSize 0

Context: server config, virtual host

Status: Extension

Module: mod_proxy

The `ProxyReceiveBufferSize` directive specifies an explicit (TCP/IP) network buffer size for proxied HTTP and FTP connections, for increased throughput. It has to be greater than 512 or set to 0 to indicate that the system's default buffer size should be used.

Example

```
ProxyReceiveBufferSize 2048
```



Description:	Remote proxy used to handle certain requests
Syntax:	ProxyRemote <i>match remote-server</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This defines remote proxies to this proxy. *match* is either the name of a URL-scheme that the remote server supports, or a partial URL for which the remote server should be used, or * to indicate the server should be contacted for all requests. *remote-server* is a partial URL for the remote server. Syntax:

```
remote-server = scheme://hostname[:port]
```

scheme is effectively the protocol that should be used to communicate with the remote server; only `http` and `https` are supported by this module. When using `https`, the requests are forwarded through the remote proxy using the HTTP CONNECT method.

Example

```
ProxyRemote http://goodguys.example.com/  
http://mirrorguys.example.com:8000  
ProxyRemote * http://cleverproxy.localdomain  
ProxyRemote ftp http://ftpproxy.mydomain:8080
```

In the last example, the proxy will forward FTP requests, encapsulated as yet another HTTP proxy request, to another proxy which can handle them.

This option also supports reverse proxy configuration; a backend webserver can be embedded within a virtualhost URL space even if that server is hidden by another forward proxy.



ProxyRemoteMatch Directive

Description:	Remote proxy used to handle requests matched by regular expressions
Syntax:	<code>ProxyRemoteMatch <i>regex remote-server</i></code>
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

The `ProxyRemoteMatch` is identical to the `ProxyRemote` directive, except that the first argument is a [regular expression](#) match against the requested URL.



Description:	Enables forward (standard) proxy requests
Syntax:	ProxyRequests On Off
Default:	ProxyRequests Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This allows or prevents Apache from functioning as a forward proxy server. (Setting ProxyRequests to Off does not disable use of the [ProxyPass](#) directive.)

In a typical reverse proxy or gateway configuration, this option should be set to Off.

In order to get the functionality of proxying HTTP or FTP sites, you need also [mod_proxy_http](#) or [mod_proxy_ftp](#) (or both) present in the server.

In order to get the functionality of (forward) proxying HTTPS sites, you need [mod_proxy_connect](#) enabled in the server.

Warning

Do not enable proxying with [ProxyRequests](#) until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

See also

- [Forward and Reverse Proxies/Gateways](#)



Description:	Set various Proxy balancer or member parameters
Syntax:	<code>ProxySet <i>url</i> <i>key=value</i> [<i>key=value ...</i>]</code>
Context:	directory
Status:	Extension
Module:	mod_proxy
Compatibility:	ProxySet is only available in Apache 2.2 and later.

This directive is used as an alternate method of setting any of the parameters available to Proxy balancers and workers normally done via the [ProxyPass](#) directive. If used within a `<Proxy balancer url|worker url>` container directive, the *url* argument is not required. As a side effect the respective balancer or worker gets created. This can be useful when doing reverse proxying via a [RewriteRule](#) instead of a [ProxyPass](#) directive.

```
<Proxy balancer://hotcluster>
  BalancerMember http://www2.example.com:8080 loadfactor=1
  BalancerMember http://www3.example.com:8080 loadfactor=2
  ProxySet lbmethod=bytraffic
</Proxy>
```

```
<Proxy http://backend>
  ProxySet keepalive=On
</Proxy>
```

```
ProxySet balancer://foo lbmethod=bytraffic timeout=15
```

```
ProxySet ajp://backend:7001 timeout=15
```

Warning

Keep in mind that the same parameter key can have a different meaning depending whether it is applied to a balancer or a worker, as shown by the two examples above regarding timeout.



ProxyStatus Directive

Description:	Show Proxy LoadBalancer status in mod_status
Syntax:	ProxyStatus Off On Full
Default:	ProxyStatus Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in version 2.2 and later

This directive determines whether or not proxy loadbalancer status data is displayed via the [mod_status](#) server-status page.

Note

Full is synonymous with **On**



ProxyTimeout Directive

Description:	Network timeout for proxied requests
Syntax:	ProxyTimeout <i>seconds</i>
Default:	Value of Timeout
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in Apache 2.0.31 and later

This directive allows a user to specify a timeout on proxy requests. This is useful when you have a slow/buggy appserver which hangs, and you would rather just return a timeout and fail gracefully instead of waiting however long it takes the server to return.



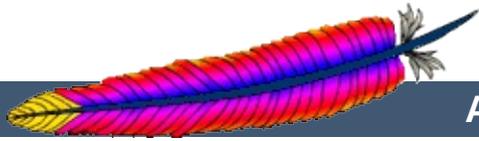
Description:	Information provided in the Via HTTP response header for proxied requests
Syntax:	ProxyVia On Off Full Block
Default:	ProxyVia Off
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This directive controls the use of the `Via: HTTP` header by the proxy. Its intended use is to control the flow of proxy requests along a chain of proxy servers. See [RFC 2616](#) (HTTP/1.1), section 14.45 for an explanation of `Via: header lines`.

- If set to `Off`, which is the default, no special processing is performed. If a request or reply contains a `Via: header`, it is passed through unchanged.
- If set to `On`, each request and reply will get a `Via: header line` added for the current host.
- If set to `Full`, each generated `Via: header line` will additionally have the Apache server version shown as a `Via: comment field`.
- If set to `Block`, every proxy request will have all its `Via: header lines` removed. No new `Via: header` will be generated.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_proxy_ajp`

Description:	AJP support module for mod_proxy
Status:	Extension
Module Identifier:	<code>proxy_ajp_module</code>
Source File:	<code>mod_proxy_ajp.c</code>
Compatibility:	Available in version 2.1 and later

Summary

This module *requires* the service of [mod_proxy](#). It provides support for the Apache JServ Protocol version 1.3 (hereafter *AJP13*).

Thus, in order to get the ability of handling AJP13 protocol, [mod_proxy](#) and [mod_proxy_ajp](#) have to be present in the server.

Warning

Do not enable proxying until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

See also

[mod_proxy](#)

[Environment Variable documentation](#)



This module is used to reverse proxy to a backend application server (e.g. Apache Tomcat) using the AJP13 protocol. The usage is similar to an HTTP reverse proxy, but uses the `ajp://` prefix:

Simple Reverse Proxy

```
ProxyPass /app ajp://backend.example.com:8009/app
```

Balancers may also be used:

Balancer Reverse Proxy

```
<Proxy balancer://cluster>
  BalancerMember ajp://app1.example.com:8009 loadfactor=1
  BalancerMember ajp://app2.example.com:8009 loadfactor=2
  ProxySet lbmethod=bytraffic
</Proxy>
ProxyPass /app balancer://cluster/app
```

Note that usually no [ProxyPassReverse](#) directive is necessary. The AJP request includes the original host header given to the proxy, and the application server can be expected to generate self-referential headers relative to this host, so no rewriting is necessary.

The main exception is when the URL path on the proxy differs from that on the backend. In this case, a redirect header can be rewritten relative to the original host URL (not the backend `ajp://` URL), for example:

Rewriting Proxied Path

```
ProxyPass /apps/foo ajp://backend.example.com:8009/foo
ProxyPassReverse /apps/foo http://www.example.com/foo
```

However, it is usually better to deploy the application on the backend server at the same path as the proxy rather than to take this approach.



Environment Variables

Environment variables whose names have the prefix `AJP_` are forwarded to the origin server as AJP request attributes (with the `AJP_` prefix removed from the name of the key).



Structure of the Protocol

The AJP13 protocol is packet-oriented. A binary format was presumably chosen over the more readable plain text for reasons of performance. The web server communicates with the servlet container over TCP connections. To cut down on the expensive process of socket creation, the web server will attempt to maintain persistent TCP connections to the servlet container, and to reuse a connection for multiple request/response cycles.

Once a connection is assigned to a particular request, it will not be used for any others until the request-handling cycle has terminated. In other words, requests are not multiplexed over connections. This makes for much simpler code at either end of the connection, although it does cause more connections to be open at once.

Once the web server has opened a connection to the servlet container, the connection can be in one of the following states:

- Idle
No request is being handled over this connection.
- Assigned
The connection is handling a specific request.

Once a connection is assigned to handle a particular request, the basic request information (e.g. HTTP headers, etc) is sent over the connection in a highly condensed form (e.g. common strings are encoded as integers). Details of that format are below in Request Packet Structure. If there is a body to the request (`content-length > 0`), that is sent in a separate packet immediately after.

At this point, the servlet container is presumably ready to start processing the request. As it does so, it can send the following messages back to the web server:

- SEND_HEADERS

- Send a set of headers back to the browser.
- SEND_BODY_CHUNK
Send a chunk of body data back to the browser.
 - GET_BODY_CHUNK
Get further data from the request if it hasn't all been transferred yet. This is necessary because the packets have a fixed maximum size and arbitrary amounts of data can be included the body of a request (for uploaded files, for example). (Note: this is unrelated to HTTP chunked transfer).
 - END_RESPONSE
Finish the request-handling cycle.

Each message is accompanied by a differently formatted packet of data. See Response Packet Structures below for details.



Basic Packet Structure

There is a bit of an XDR heritage to this protocol, but it differs in lots of ways (no 4 byte alignment, for example).

AJP13 uses network byte order for all data types.

There are four data types in the protocol: bytes, booleans, integers and strings.

Byte

A single byte.

Boolean

A single byte, `1 = true`, `0 = false`. Using other non-zero values as true (i.e. C-style) may work in some places, but it won't in others.

Integer

A number in the range of `0` to 2^{16} (32768). Stored in 2 bytes with the high-order byte first.

String

A variable-sized string (length bounded by 2^{16}). Encoded with the length packed into two bytes first, followed by the string (including the terminating `\0`). Note that the encoded length does **not** include the trailing `\0` -- it is like `strlen`. This is a touch confusing on the Java side, which is littered with odd autoincrement statements to skip over these terminators. I believe the reason this was done was to allow the C code to be extra efficient when reading strings which the servlet container is sending back -- with the terminating `\0` character, the C code can pass around references into a single buffer, without copying. if the `\0` was missing, the C code would have to copy things out in order to get its notion of a string.

Packet Size

According to much of the code, the max packet size is $8 * 1024$ bytes (8K). The actual length of the packet is encoded in the header.

Packet Headers

Packets sent from the server to the container begin with 0x1234. Packets sent from the container to the server begin with AB (that's the ASCII code for A followed by the ASCII code for B). After those first two bytes, there is an integer (encoded as above) with the length of the payload. Although this might suggest that the maximum payload could be as large as 2^{16} , in fact, the code sets the maximum to be 8K.

Packet Format (Server->Container)					
Byte	0	1	2	3	4...(n+3)
Contents	0x12	0x34	Data	Length (n)	Data

Packet Format (Container->Server)					
Byte	0	1	2	3	4...(n+3)
Contents	A	B	Data	Length (n)	Data

For most packets, the first byte of the payload encodes the type of message. The exception is for request body packets sent from the server to the container -- they are sent with a standard packet header (0x1234 and then length of the packet), but without any prefix code after that.

The web server can send the following messages to the servlet container:

Code	Type of Packet	Meaning
2	Forward Request	Begin the request-processing cycle with the following data
7	Shutdown	The web server asks the container to shut itself down.
8	Ping	The web server asks the container to take control (secure login phase).
10	CPing	The web server asks the container to respond

none	Data	quickly with a CPong. Size (2 bytes) and corresponding body data.
------	------	--

To ensure some basic security, the container will only actually do the Shut down if the request comes from the same machine on which it's hosted.

The first Data packet is send immediately after the Forward Request by the web server.

The servlet container can send the following types of messages to the webserver:

Code	Type of Packet	Meaning
3	Send Body Chunk	Send a chunk of the body from the servlet container to the web server (and presumably, onto the browser).
4	Send Headers	Send the response headers from the servlet container to the web server (and presumably, onto the browser).
5	End Response	Marks the end of the response (and thus the request-handling cycle).
6	Get Body Chunk	Get further data from the request if it hasn't all been transferred yet.
9	CPong Reply	The reply to a CPing request

Each of the above messages has a different internal structure, detailed below.



For messages from the server to the container of type *Forward Request*:

```
AJP13_FORWARD_REQUEST :=
  prefix_code      (byte) 0x02 = JK_AJP13_FORWARD_REQUEST
  method          (byte)
  protocol        (string)
  req_uri         (string)
  remote_addr     (string)
  remote_host     (string)
  server_name     (string)
  server_port     (integer)
  is_ssl         (boolean)
  num_headers     (integer)
  request_headers *(req_header_name req_header_value)
  attributes      *(attribut_name attribute_value)
  request_terminator (byte) 0xFF
```

The `request_headers` have the following structure:

```
req_header_name :=
  sc_req_header_name | (string) [see below for how this is parsed]
sc_req_header_name := 0xA0xx (integer)
req_header_value := (string)
```

The `attributes` are optional and have the following structure:

```
attribute_name := sc_a_name | (sc_a_req_attribute string)
attribute_value := (string)
```

Not that the all-important header is `content-length`, because it determines whether or not the container looks for another packet immediately.

Detailed description of the elements of Forward Request

Request prefix

For all requests, this will be 2. See above for details on other Prefix codes.

Method

The HTTP method, encoded as a single byte:

Command Name	Code
OPTIONS	1
GET	2
HEAD	3
POST	4
PUT	5
DELETE	6
TRACE	7
PROPFIND	8
PROPPATCH	9
MKCOL	10
COPY	11
MOVE	12
LOCK	13
UNLOCK	14
ACL	15
REPORT	16
VERSION-CONTROL	17
CHECKIN	18
CHECKOUT	19
UNCHECKOUT	20
SEARCH	21
MKWORKSPACE	22
UPDATE	23

LABEL	24
MERGE	25
BASELINE_CONTROL	26
MKACTIVITY	27

Later version of ajp13, will transport additional methods, even if they are not in this list.

protocol, req_uri, remote_addr, remote_host, server_name, server_port, is_ssl

These are all fairly self-explanatory. Each of these is required, and will be sent for every request.

Headers

The structure of request_headers is the following: First, the number of headers num_headers is encoded. Then, a series of header name req_header_name / value req_header_value pairs follows. Common header names are encoded as integers, to save space. If the header name is not in the list of basic headers, it is encoded normally (as a string, with prefixed length). The list of common headers sc_req_header_name and their codes is as follows (all are case-sensitive):

Name	Code value	Code name
accept	0xA001	SC_REQ_ACCEPT
accept-charset	0xA002	SC_REQ_ACCEPT_CHARSET
accept-encoding	0xA003	SC_REQ_ACCEPT_ENCODING
accept-language	0xA004	SC_REQ_ACCEPT_LANGUAGE
authorization	0xA005	SC_REQ_AUTHORIZATION
connection	0xA006	SC_REQ_CONNECTION
content-type	0xA007	SC_REQ_CONTENT_TYPE

content-length	0xA008	SC_REQ_CONTENT_LENGTH
cookie	0xA009	SC_REQ_COOKIE
cookie2	0xA00A	SC_REQ_COOKIE2
host	0xA00B	SC_REQ_HOST
pragma	0xA00C	SC_REQ_PRAGMA
referer	0xA00D	SC_REQ_REFERER
user-agent	0xA00E	SC_REQ_USER_AGENT

The Java code that reads this grabs the first two-byte integer and if it sees an '0xA0' in the most significant byte, it uses the integer in the second byte as an index into an array of header names. If the first byte is not 0xA0, it assumes that the two-byte integer is the length of a string, which is then read in.

This works on the assumption that no header names will have length greater than 0x9FFF (==0xA000 - 1), which is perfectly reasonable, though somewhat arbitrary.

Note:

The content-length header is extremely important. If it is present and non-zero, the container assumes that the request has a body (a POST request, for example), and immediately reads a separate packet off the input stream to get that body.

Attributes

The attributes prefixed with a ? (e.g. ?context) are all optional. For each, there is a single byte code to indicate the type of attribute, and then its value (string or integer). They can be sent in any order (though the C code always sends them in the order listed below). A special terminating code is sent to signal the end of the list of optional attributes. The list of byte codes is:

--

Information	Code Value	Type Of Value	Note
?context	0x01	-	Not currently implemented
? servlet_path	0x02	-	Not currently implemented
? remote_user	0x03	String	
?auth_type	0x04	String	
? query_string	0x05	String	
?jvm_route	0x06	String	
?ssl_cert	0x07	String	
?ssl_cipher	0x08	String	
? ssl_session	0x09	String	
? req_attribute	0x0A	String	Name (the name of the attribute follows)
? ssl_key_size	0x0B	Integer	
are_done	0xFF	-	request_terminator

The `context` and `servlet_path` are not currently set by the C code, and most of the Java code completely ignores whatever is sent over for those fields (and some of it will actually break if a string is sent along after one of those codes). I don't know if this is a bug or an unimplemented feature or just vestigial code, but it's missing from both sides of the connection.

The `remote_user` and `auth_type` presumably refer to HTTP-level authentication, and communicate the remote user's username and the type of authentication used to establish their identity (e.g. Basic, Digest).

The `query_string`, `ssl_cert`, `ssl_cipher`, and `ssl_session`

refer to the corresponding pieces of HTTP and HTTPS.

The `jvm_route`, is used to support sticky sessions -- associating a user's session with a particular Tomcat instance in the presence of multiple, load-balancing servers.

Beyond this list of basic attributes, any number of other attributes can be sent via the `req_attribute` code `0x0A`. A pair of strings to represent the attribute name and value are sent immediately after each instance of that code. Environment values are passed in via this method.

Finally, after all the attributes have been sent, the attribute terminator, `0xFF`, is sent. This signals both the end of the list of attributes and also then end of the Request Packet.



for messages which the container can send back to the server.

```
AJP13_SEND_BODY_CHUNK :=
  prefix_code    3
  chunk_length  (integer)
  chunk          *(byte)
  chunk_terminator (byte) 0x00

AJP13_SEND_HEADERS :=
  prefix_code      4
  http_status_code (integer)
  http_status_msg  (string)
  num_headers      (integer)
  response_headers *(res_header_name header_value)

res_header_name :=
  sc_res_header_name | (string) [see below for how this is parsed]

sc_res_header_name := 0xA0 (byte)

header_value := (string)

AJP13_END_RESPONSE :=
  prefix_code  5
  reuse        (boolean)

AJP13_GET_BODY_CHUNK :=
  prefix_code      6
  requested_length (integer)
```

Details:

Send Body Chunk

The chunk is basically binary data, and is sent directly back to the browser.

Send Headers

The status code and message are the usual HTTP things (e.g. 200 and OK). The response header names are encoded the same way the request header names are. See `header_encoding` above for details about how

the codes are distinguished from the strings.
The codes for common headers are:

Name	Code value
Content-Type	0xA001
Content-Language	0xA002
Content-Length	0xA003
Date	0xA004
Last-Modified	0xA005
Location	0xA006
Set-Cookie	0xA007
Set-Cookie2	0xA008
Servlet-Engine	0xA009
Status	0xA00A
WWW-Authenticate	0xA00B

After the code or the string header name, the header value is immediately encoded.

End Response

Signals the end of this request-handling cycle. If the reuse flag is true (anything other than 0 in the actual C code), this TCP connection can now be used to handle new incoming requests. If reuse is false (==0), the connection should be closed.

Get Body Chunk

The container asks for more data from the request (If the body was too large to fit in the first packet sent over or when the request is chunked). The server will send a body packet back with an amount of data which is the minimum of the request_length, the maximum send body size (8186 (8 Kbytes - 6)), and the number of bytes actually left to

send from the request body.

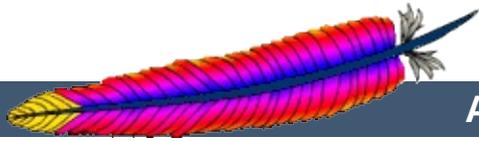
If there is no more data in the body (i.e. the servlet container is trying to read past the end of the body), the server will send back an *empty* packet, which is a body packet with a payload length of 0.

(0x12, 0x34, 0x00, 0x00)

Copyright 2017 The Apache Software Foundation.

Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_proxy_balancer`

Description:	<code>mod_proxy</code> extension for load balancing
Status:	Extension
Module Identifier:	<code>proxy_balancer_module</code>
Source File:	<code>mod_proxy_balancer.c</code>
Compatibility:	Available in version 2.1 and later

Summary

This module *requires* the service of `mod_proxy`. It provides load balancing support for HTTP, FTP and AJP13 protocols

Thus, in order to get the ability of load balancing, `mod_proxy` and `mod_proxy_balancer` have to be present in the server.

Warning

Do not enable proxying until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

See also

[mod_proxy](#)



At present, there are 3 load balancer scheduler algorithms available for use: Request Counting, Weighted Traffic Counting and Pending Request Counting. These are controlled via the `lbmethod` value of the Balancer definition. See the [ProxyPass](#) directive for more information.



The balancer supports stickyness. When a request is proxied to some back-end, then all following requests from the same user should be proxied to the same back-end. Many load balancers implement this feature via a table that maps client IP addresses to back-ends. This approach is transparent to clients and back-ends, but suffers from some problems: unequal load distribution if clients are themselves hidden behind proxies, stickyness errors when a client uses a dynamic IP address that changes during a session and loss of stickyness, if the mapping table overflows.

The module [mod_proxy_balancer](#) implements stickyness on top of two alternative means: cookies and URL encoding. Providing the cookie can be either done by the back-end or by the Apache web server itself. The URL encoding is usually done on the back-end.



Example of a balancer configuration

Before we dive into the technical details, here's an example of how you might use [mod_proxy_balancer](#) to provide load balancing between two back-end servers:

```
<Proxy balancer://mycluster>
BalancerMember http://192.168.1.50:80
BalancerMember http://192.168.1.51:80
</Proxy>
ProxyPass /test balancer://mycluster
```

Another example of how to provide load balancing with stickyness using [mod_headers](#), even if the back-end server does not set a suitable session cookie:

```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/"
env=BALANCER_ROUTE_CHANGED
<Proxy balancer://mycluster>
BalancerMember http://192.168.1.50:80 route=1
BalancerMember http://192.168.1.51:80 route=2
ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass /test balancer://mycluster
```



Request Counting Algorithm

Enabled via `lbmethod=byrequests`, the idea behind this scheduler is that we distribute the requests among the various workers to ensure that each gets their configured share of the number of requests. It works as follows:

lbfactor is how much we expect this worker to work, or the workers's work quota. This is a normalized value representing their "share" of the amount of work to be done.

lbstatus is how urgent this worker has to work to fulfill its quota of work.

The *worker* is a member of the load balancer, usually a remote host serving one of the supported protocols.

We distribute each worker's work quota to the worker, and then look which of them needs to work most urgently (biggest *lbstatus*). This worker is then selected for work, and its *lbstatus* reduced by the total work quota we distributed to all workers. Thus the sum of all *lbstatus* does not change(*) and we distribute the requests as desired.

If some workers are disabled, the others will still be scheduled correctly.

```
for each worker in workers
  worker lbstatus += worker lbfactor
  total factor    += worker lbfactor
  if worker lbstatus > candidate lbstatus
    candidate = worker

candidate lbstatus -= total factor
```

If a balancer is configured as follows:

worker	a	b	c	d
lbfactor	25	25	25	25
lbstatus	0	0	0	0

And *b* gets disabled, the following schedule is produced:

worker	a	b	c	d
lbstatus	-50	0	25	25
lbstatus	-25	0	-25	50
lbstatus	0	0	0	0
			(repeat)	

That is it schedules: *a c d a c d a c d ...* Please note that:

worker	a	b	c	d
lbfactor	25	25	25	25

Has the exact same behavior as:

worker	a	b	c	d
lbfactor	1	1	1	1

This is because all values of *lbfactor* are normalized with respect to the others. For:

worker	a	b	c
lbfactor	1	4	1

worker *b* will, on average, get 4 times the requests that *a* and *c* will.

The following asymmetric configuration works as one would expect:

worker	a	b
lbfactor	70	30
lbstatus	-30	30
lbstatus	40	-40
lbstatus	10	-10

lbstatus	-20	20
lbstatus	-50	50
lbstatus	20	-20
lbstatus	-10	10
lbstatus	-40	40
lbstatus	30	-30
lbstatus	0	0
	(repeat)	

That is after 10 schedules, the schedule repeats and 7 *a* are selected with 3 *b* interspersed.



Weighted Round Robin Algorithm

Enabled via `lbmethod=bytraffic`, the idea behind this scheduler is very similar to the Request Counting method, with the following changes:

lbfactor is how much traffic, in bytes, we want this worker to handle. This is also a normalized value representing their "share" of the amount of work to be done, but instead of simply counting the number of requests, we take into account the amount of traffic this worker has seen.

If a balancer is configured as follows:

worker	a	b	c
lbfactor	1	2	1

Then we mean that we want *b* to process twice the amount of bytes than *a* or *c* should. It does not necessarily mean that *b* would handle twice as many requests, but it would process twice the I/O. Thus, the size of the request and response are applied to the weighting and selection algorithm.



Request Counting Algorithm

Enabled via `lbmethod=bybusyness`, this scheduler keeps track of how many requests each worker is assigned at present. A new request is automatically assigned to the worker with the lowest number of active requests. This is useful in the case of workers that queue incoming requests independently of Apache, to ensure that queue length stays even and a request is always given to the worker most likely to service it fastest.

In the case of multiple least-busy workers, the statistics (and weightings) used by the Request Counting method are used to break the tie. Over time, the distribution of work will come to resemble that characteristic of `byrequests`.

This algorithm is available in Apache HTTP Server 2.2.10 and later.



At present there are 6 environment variables exported:

BALANCER_SESSION_STICKY

This is assigned the *stickysession* value used for the current request. It is the name of the cookie or request parameter used for sticky sessions

BALANCER_SESSION_ROUTE

This is assigned the *route* parsed from the current request.

BALANCER_NAME

This is assigned the name of the balancer used for the current request. The value is something like `balancer://foo`.

BALANCER_WORKER_NAME

This is assigned the name of the worker used for the current request. The value is something like `http://hostA:1234`.

BALANCER_WORKER_ROUTE

This is assigned the *route* of the worker that will be used for the current request.

BALANCER_ROUTE_CHANGED

This is set to 1 if the session route does not match the worker route (`BALANCER_SESSION_ROUTE != BALANCER_WORKER_ROUTE`) or the session does not yet have an established route. This can be used to determine when/if the client needs to be sent an updated route when sticky sessions are used.



This module *requires* the service of `mod_status`. Balancer manager enables dynamic update of balancer members. You can use balancer manager to change the balance factor of a particular member, or put it in the off line mode.

Thus, in order to get the ability of load balancer management, `mod_status` and `mod_proxy_balancer` have to be present in the server.

To enable load balancer management for browsers from the example.com domain add this code to your `httpd.conf` configuration file

```
<Location /balancer-manager>
  SetHandler balancer-manager

  Order Deny,Allow
  Deny from all
  Allow from .example.com
</Location>
```

You can now access load balancer manager by using a Web browser to access the page `http://your.server.name/balancer-manager`



COOKIE BASED STICKYNESS

When using cookie based stickyness, you need to configure the name of the cookie that contains the information about which back-end to use. This is done via the *stickysession* attribute added to either [ProxyPass](#) or [ProxySet](#). The name of the cookie is case-sensitive. The balancer extracts the value of the cookie and looks for a member worker with *route* equal to that value. The *route* must also be set in either [ProxyPass](#) or [ProxySet](#). The cookie can either be set by the back-end, or as shown in the above [example](#) by the Apache web server itself.

Some back-ends use a slightly different form of stickyness cookie, for instance Apache Tomcat. Tomcat adds the name of the Tomcat instance to the end of its session id cookie, separated with a dot (.) from the session id. Thus if the Apache web server finds a dot in the value of the stickyness cookie, it only uses the part behind the dot to search for the route. In order to let Tomcat know about its instance name, you need to set the attribute `jvmRoute` inside the Tomcat configuration file `conf/server.xml` to the value of the *route* of the worker that connects to the respective Tomcat. The name of the session cookie used by Tomcat (and more generally by Java web applications based on servlets) is `JSESSIONID` (upper case) but can be configured to something else.

The second way of implementing stickyness is URL encoding. The web server searches for a query parameter in the URL of the request. The name of the parameter is specified again using *stickysession*. The value of the parameter is used to lookup a member worker with *route* equal to that value. Since it is not easy to extract and manipulate all URL links contained in responses, generally the work of adding the parameters to each link is done by the back-end generating the content. In some cases it might be feasible doing this via the web server using [mod_substitute](#). This can have negative impact on performance though.

The Java standards implement URL encoding slightly different. They use

a path info appended to the URL using a semicolon (;) as the separator and add the session id behind. As in the cookie case, Apache Tomcat can include the configured `JvmRoute` in this path info. To let Apache find this sort of path info, you need to set `sColonPathDelim` to `On` in [ProxyPass](#) or [ProxySet](#).

Finally you can support cookies and URL encoding at the same time, by configuring the name of the cookie and the name of the URL parameter separated by a vertical bar (|) as in the following example:

```
ProxyPass /test balancer://mycluster
stickySession=JSESSIONID|jsessionId sColonPathDelim=On
<Proxy balancer://mycluster>
BalancerMember http://192.168.1.50:80 route=node1
BalancerMember http://192.168.1.51:80 route=node2
</Proxy>
```

If the cookie and the request parameter both provide routing information for the same request, the information from the request parameter is used.



If you experience stickyness errors, e.g. users lose their application sessions and need to login again, you first want to check whether this is because the back-ends are sometimes unavailable or whether your configuration is wrong. To find out about possible stability problems with the back-ends, check your Apache error log for proxy error messages.

To verify your configuration, first check, whether the stickyness is based on a cookie or on URL encoding. Next step would be logging the appropriate data in the access log by using an enhanced [LogFormat](#). The following fields are useful:

%{MYCOOKIE}C

The value contained in the cookie with name MYCOOKIE. The name should be the same given in the *stickysession* attribute.

%{Set - Cookie}o

This logs any cookie set by the back-end. You can track, whether the back-end sets the session cookie you expect, and to which value it is set.

%{BALANCER_SESSION_STICKY}e

The name of the cookie or request parameter used to lookup the routing information.

%{BALANCER_SESSION_ROUTE}e

The route information found in the request.

%{BALANCER_WORKER_ROUTE}e

The route of the worker chosen.

%{BALANCER_ROUTE_CHANGED}e

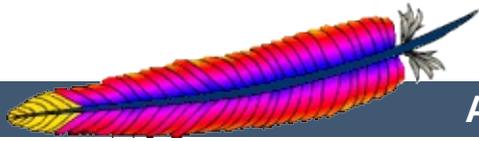
Set to 1 if the route in the request is different from the route of the worker, i.e. the request couldn't be handled sticky.

Common reasons for loss of session are session timeouts, which are usually configurable on the back-end server.

The balancer also logs detailed information about handling stickyness to the error log, if the log level is set to debug or higher. This is an easy way to troubleshoot stickyness problems, but the log volume might be too high for production servers under high load.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_proxy_connect`

Description:	<code>mod_proxy</code> extension for CONNECT request handling
Status:	Extension
Module Identifier:	<code>proxy_connect_module</code>
Source File:	<code>mod_proxy_connect.c</code>

Summary

This module *requires* the service of `mod_proxy`. It provides support for the CONNECT HTTP method. This method is mainly used to tunnel SSL requests through proxy servers.

Thus, in order to get the ability of handling CONNECT requests, `mod_proxy` and `mod_proxy_connect` have to be present in the server.

CONNECT is also used, when the server needs to send an HTTPS request through a forward proxy. In this case the server acts as a CONNECT client. This functionality is part of `mod_proxy` and `mod_proxy_connect` is not needed in this case.

Warning

Do not enable proxying until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

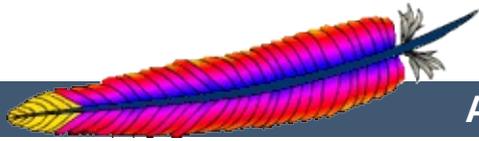
See also

[AllowCONNECT](#)

[mod_proxy](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_proxy_ftp`

Description:	FTP support module for mod_proxy
Status:	Extension
Module Identifier:	<code>proxy_ftp_module</code>
Source File:	<code>mod_proxy_ftp.c</code>

Summary

This module *requires* the service of [mod_proxy](#). It provides support for the proxying FTP sites. Note that FTP support is currently limited to the GET method.

Thus, in order to get the ability of handling FTP proxy requests, [mod_proxy](#) and [mod_proxy_ftp](#) have to be present in the server.

Warning

Do not enable proxying until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

See also

[mod_proxy](#)



You probably don't have that particular file type defined as `application/octet-stream` in your proxy's `mime.types` configuration file. A useful line can be

```
application/octet-stream  bin dms lha lzh exe class tgz taz
```

Alternatively you may prefer to default everything to binary:

```
DefaultType application/octet-stream
```



In the rare situation where you must download a specific file using the FTP ASCII transfer method (while the default transfer is in binary mode), you can override `mod_proxy`'s default by suffixing the request with `;type=a` to force an ASCII transfer. (FTP Directory listings are always executed in ASCII mode, however.)



How can we do HTTP uploads?

Currently, only GET is supported for FTP in mod_proxy. You can of course use HTTP upload (POST or PUT) through an Apache proxy.



directory?

An FTP URI is interpreted relative to the home directory of the user who is logging in. Alas, to reach higher directory levels you cannot use `../`, as the dots are interpreted by the browser and not actually sent to the FTP server. To address this problem, the so called *Squid %2f hack* was implemented in the Apache FTP proxy; it is a solution which is also used by other popular proxy servers like the [Squid Proxy Cache](#). By prepending `/%2f` to the path of your request, you can make such a proxy change the FTP starting directory to `/` (instead of the home directory). For example, to retrieve the file `/etc/motd`, you would use the URL:

```
ftp://user@host/%2f/etc/motd
```



browser's URL line?

To log in to an FTP server by username and password, Apache uses different strategies. In absence of a user name and password in the URL altogether, Apache sends an anonymous login to the FTP server, *i.e.*,

```
user: anonymous  
password: apache_proxy@
```

This works for all popular FTP servers which are configured for anonymous access.

For a personal login with a specific username, you can embed the user name into the URL, like in:

```
ftp://username@host/myfile
```

If the FTP server asks for a password when given this username (which it should), then Apache will reply with a 401 (Authorization required) response, which causes the Browser to pop up the username/password dialog. Upon entering the password, the connection attempt is retried, and if successful, the requested resource is presented. The advantage of this procedure is that your browser does not display the password in cleartext (which it would if you had used

```
ftp://username:password@host/myfile
```

in the first place).

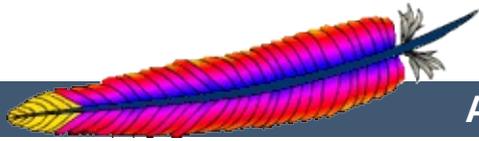
Note

The password which is transmitted in such a way is not encrypted on its way. It travels between your browser and the Apache proxy server in a base64-encoded cleartext string, and between the Apache proxy and the FTP server as plaintext. You should therefore think twice

before accessing your FTP server via HTTP (or before accessing your personal files via FTP at all!) When using insecure channels, an eavesdropper might intercept your password on its way.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_proxy_http`

Description:	HTTP support module for mod_proxy
Status:	Extension
Module Identifier:	<code>proxy_http_module</code>
Source File:	<code>mod_proxy_http.c</code>

Summary

This module *requires* the service of [mod_proxy](#). It provides the features used for proxying HTTP and HTTPS requests. [mod_proxy_http](#) supports HTTP/0.9, HTTP/1.0 and HTTP/1.1. It does *not* provide any caching abilities. If you want to set up a caching proxy, you might want to use the additional service of the [mod_cache](#) module.

Thus, in order to get the ability of handling HTTP proxy requests, [mod_proxy](#) and [mod_proxy_http](#) have to be present in the server.

Warning

Do not enable proxying until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

See also

[mod_proxy](#)

[mod_proxy_connect](#)



In addition to the configuration directives that control the behaviour of `mod_proxy`, there are a number of *environment variables* that control the HTTP protocol provider. Environment variables below that don't specify specific values are enabled when set to any value.

proxy-sendextracrlf

Causes proxy to send an extra CR-LF newline on the end of a request. This is a workaround for a bug in some browsers.

force-proxy-request-1.0

Forces the proxy to send requests to the backend as HTTP/1.0 and disables HTTP/1.1 features.

proxy-nokeepalive

Forces the proxy to close the backend connection after each request.

proxy-chain-auth

If the proxy requires authentication, it will read and consume the proxy authentication credentials sent by the client. With *proxy-chain-auth* it will *also* forward the credentials to the next proxy in the chain. This may be necessary if you have a chain of proxies that share authentication information. **Security Warning:** Do not set this unless you know you need it, as it forwards sensitive information!

proxy-sendcl

HTTP/1.0 required all HTTP requests that include a body (e.g. POST requests) to include a *Content-Length* header. This environment variable forces the Apache proxy to send this header to the backend server, regardless of what the Client sent to the proxy. It ensures compatibility when proxying for an HTTP/1.0 or unknown backend. However, it may require the entire request to be buffered by the proxy, so it becomes very inefficient for large requests.

proxy-sendchunks or proxy-sendchunked

This is the opposite of *proxy-sendcl*. It allows request bodies to be

sent to the backend using chunked transfer encoding. This allows the request to be efficiently streamed, but requires that the backend server supports HTTP/1.1.

proxy-interim-response

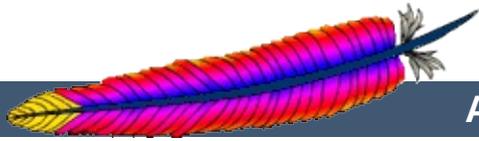
This variable takes values RFC or Suppress. Earlier httpd versions would suppress HTTP interim (1xx) responses sent from the backend. This is technically a violation of the HTTP protocol. In practice, if a backend sends an interim response, it may itself be extending the protocol in a manner we know nothing about, or just broken. So this is now configurable: set `proxy-interim-response RFC` to be fully protocol compliant, or `proxy-interim-response Suppress` to suppress interim responses.

proxy-initial-not-pooled

If this variable is set no pooled connection will be reused if the client connection is an initial connection. This avoids the "proxy: error reading status line from remote server" error message caused by the race condition that the backend server closed the pooled connection after the connection check by the proxy and before data sent by the proxy reached the backend. It has to be kept in mind that setting this variable downgrades performance, especially with HTTP/1.0 clients.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_proxy_scgi`

Description:	SCGI gateway module for mod_proxy
Status:	Extension
Module Identifier:	<code>proxy_scgi_module</code>
Source File:	<code>mod_proxy_scgi.c</code>
Compatibility:	Available in version 2.2.14 and later

Summary

This module *requires* the service of [mod_proxy](#). It provides support for the [SCGI protocol, version 1](#).

Thus, in order to get the ability of handling the SCGI protocol, [mod_proxy](#) and [mod_proxy_scgi](#) have to be present in the server.

Warning

Do not enable proxying until you have [secured your server](#). Open proxy servers are dangerous both to your network and to the Internet at large.

See also

[mod_proxy](#)

[mod_proxy_balancer](#)



Remember, in order to make the following examples work, you have to enable [mod_proxy](#) and [mod_proxy_scgi](#).

Simple gateway

```
ProxyPass /scgi-bin/ scgi://localhost:4000/
```

The balanced gateway needs [mod_proxy_balancer](#) in addition to the already mentioned proxy modules.

Balanced gateway

```
ProxyPass /scgi-bin/ balancer://somecluster/  
<Proxy balancer://somecluster/>  
  BalancerMember scgi://localhost:4000/  
  BalancerMember scgi://localhost:4001/  
</Proxy>
```



Description:	Enable or disable internal redirect responses from the backend
Syntax:	ProxySCGIInternalRedirect On Off
Default:	ProxySCGIInternalRedirect On
Context:	server config, virtual host, directory
Status:	Extension
Module:	mod_proxy_scgi

The `ProxySCGIInternalRedirect` enables the backend to internally redirect the gateway to a different URL. This feature originates in `mod_cgi`, which internally redirects the response, if the response status is OK (200) and the response contains a `Location` header and its value starts with a slash (/). This value is interpreted as a new local URL the apache internally redirects to.

`mod_proxy_scgi` does the same as `mod_cgi` in this regard, except that you can turn off the feature.

Example

```
ProxySCGIInternalRedirect Off
```



Description:	Enable evaluation of <i>X-Sendfile</i> pseudo response header
Syntax:	<code>ProxySCGISendfile On Off Headername</code>
Default:	<code>ProxySCGISendfile Off</code>
Context:	server config, virtual host, directory
Status:	Extension
Module:	<code>mod_proxy_scgi</code>

The `ProxySCGISendfile` directive enables the SCGI backend to let files serve directly by the gateway. This is useful performance purposes -- the httpd can use `sendfile` or other optimizations, which are not possible if the file comes over the backend socket.

The `ProxySCGISendfile` argument determines the gateway behaviour:

off

No special handling takes place.

On

The gateway looks for a backend response header called `X-Sendfile` and interprets the value as filename to serve. The header is removed from the final response headers. This is equivalent to `ProxySCGISendfile X-Sendfile`.

anything else

Similar to `On`, but instead of the hardcoded header name the argument is applied as header name.

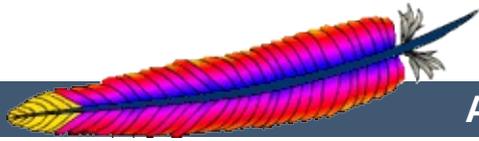
Example

```
# Use the default header (X-Sendfile)
ProxySCGISendfile On

# Use a different header
ProxySCGISendfile X-Send-Static
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_reqtimeout

Description:	Set timeout and minimum data rate for receiving requests
Status:	Extension
Module Identifier:	reqtimeout_module
Source File:	mod_reqtimeout.c
Compatibility:	Available in Apache 2.2.15 and later



1. Allow 10 seconds to receive the request including the headers and 30 seconds for receiving the request body:

```
RequestReadTimeout header=10 body=30
```

2. Allow at least 10 seconds to receive the request body. If the client sends data, increase the timeout by 1 second for every 1000 bytes received, with no upper limit for the timeout (except for the limit given indirectly by [LimitRequestBody](#)):

```
RequestReadTimeout body=10,MinRate=1000
```

3. Allow at least 10 seconds to receive the request including the headers. If the client sends data, increase the timeout by 1 second for every 500 bytes received. But do not allow more than 30 seconds for the request including the headers:

```
RequestReadTimeout header=10-30,MinRate=500
```

4. Usually, a server should have both header and body timeouts configured. If a common configuration is used for http and https virtual hosts, the timeouts should not be set too low:

```
RequestReadTimeout header=20-40,MinRate=500 body=20,MinRate=500
```



Description:	Set timeout values for receiving request headers and body from client.
Syntax:	<code>RequestReadTimeout [header=<i>timeout</i>[[<i>-maxtimeout</i>],MinRate=<i>rate</i>] [body=<i>timeout</i>[[<i>-maxtimeout</i>],MinRate=<i>rate</i>]</code>
Default:	Unset; no limit
Context:	server config, virtual host
Status:	Extension
Module:	mod_reqtimeout

This directive can set various timeouts for receiving the request headers and the request body from the client. If the client fails to send headers or body within the configured time, a 408 REQUEST TIME OUT error is sent.

For SSL virtual hosts, the header timeout values include the time needed to do the initial SSL handshake. If the user's browser is configured to query certificate revocation lists and the CRL server is not reachable, the initial SSL handshake may take a significant time until the browser gives up waiting for the CRL. Therefore the header timeout values should not be set to very low values for SSL virtual hosts. The body timeout values include the time needed for SSL renegotiation (if necessary).

When an [AcceptFilter](#) is in use (usually the case on Linux and FreeBSD), the socket is not sent to the server process before at least one byte (or the whole request for `httpready`) is received. The header timeout configured with `RequestReadTimeout` is only effective after the server process has received the socket.

For each of the two timeout types (header or body), there are three ways to specify the timeout:

- **Fixed timeout value:**

```
type=timeout
```

The time in seconds allowed for reading all of the request headers or body, respectively. A value of 0 means no limit.

- **Timeout value that is increased when data is received:**

```
type=timeout,MinRate=data_rate
```

Same as above, but whenever data is received, the timeout value is increased according to the specified minimum data rate (in bytes per second).

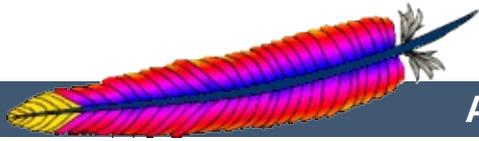
- **Timeout value that is increased when data is received, with an upper bound:**

```
type=timeout-maxtimeout,MinRate=data_rate
```

Same as above, but the timeout will not be increased above the second value of the specified timeout range.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_setenvif`

Description:	Allows the setting of environment variables based on characteristics of the request
Status:	Base
Module Identifier:	<code>setenvif_module</code>
Source File:	<code>mod_setenvif.c</code>

Summary

The `mod_setenvif` module allows you to set internal environment variables according to whether different aspects of the request match regular expressions you specify. These environment variables can be used by other parts of the server to make decisions about actions to be taken, as well as becoming available to CGI scripts and SSI pages.

The directives are considered in the order they appear in the configuration files. So more complex sequences can be used, such as this example, which sets `net_scape` if the browser is mozilla but not MSIE.

```
BrowserMatch ^Mozilla netscape
BrowserMatch MSIE !netscape
```

See also

[Environment Variables in Apache](#)



Description:	Sets environment variables conditional on HTTP User-Agent
Syntax:	BrowserMatch <i>regex</i> <i>[!]env-variable</i> <i>[=value]</i> <i>[!]env-variable</i> <i>[=value]</i> ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_setenvif

The **BrowserMatch** is a special cases of the **SetEnvIf** directive that sets environment variables conditional on the User-Agent HTTP request header. The following two lines have the same effect:

```
BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot
```

Some additional examples:

```
BrowserMatch ^Mozilla forms jpeg=yes browser=netscape
BrowserMatch "^Mozilla/[2-3]" tables agif frames javascript
BrowserMatch MSIE !javascript
```



Description:	Sets environment variables conditional on User-Agent without respect to case
Syntax:	<code>BrowserMatchNoCase <i>regex</i> [!]env-variable[=<i>value</i>] [[!]env-variable[=<i>value</i>]]</code> ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_setenvif

The `BrowserMatchNoCase` directive is semantically identical to the `BrowserMatch` directive. However, it provides for case-insensitive matching. For example:

```
BrowserMatchNoCase mac platform=macintosh
BrowserMatchNoCase win platform=windows
```

The `BrowserMatch` and `BrowserMatchNoCase` directives are special cases of the `SetEnvIf` and `SetEnvIfNoCase` directives. The following two lines have the same effect:

```
BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot
```



Description:	Sets environment variables based on attributes of the request
Syntax:	<code>SetEnvIf <i>attribute</i> <i>regex</i> [!]env-variable[=<i>value</i>] [[!]env-variable[=<i>value</i>]]</code> ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_setenvif

The `SetEnvIf` directive defines environment variables based on attributes of the request. The *attribute* specified in the first argument can be one of three things:

1. An HTTP request header field (see [RFC2616](#) for more information about these); for example: Host, User-Agent, Referer, and Accept-Language. A regular expression may be used to specify a set of request headers.
2. One of the following aspects of the request:
 - Remote_Host - the hostname (if available) of the client making the request
 - Remote_Addr - the IP address of the client making the request
 - Server_Addr - the IP address of the server on which the request was received (only with versions later than 2.0.43)
 - Request_Method - the name of the method being used (GET, POST, *et cetera*)
 - Request_Protocol - the name and version of the protocol with which the request was made (*e.g.*, "HTTP/0.9", "HTTP/1.1", *etc.*)

- Request_URI - the resource requested on the HTTP request line -- generally the portion of the URL following the scheme and host portion without the query string. See the [RewriteCond](#) directive of [mod_rewrite](#) for extra information on how to match your query string.
3. The name of an environment variable in the list of those associated with the request. This allows [SetEnvIf](#) directives to test against the result of prior matches. Only those environment variables defined by earlier [SetEnvIf \[NoCase\]](#) directives are available for testing in this manner. 'Earlier' means that they were defined at a broader scope (such as server-wide) or previously in the current directive's scope. Environment variables will be considered only if there was no match among request characteristics and a regular expression was not used for the *attribute*.

The second argument (*regex*) is a [regular expression](#). If the *regex* matches against the *attribute*, then the remainder of the arguments are evaluated.

The rest of the arguments give the names of variables to set, and optionally values to which they should be set. These take the form of

1. *varname*, or
2. *!varname*, or
3. *varname=value*

In the first form, the value will be set to "1". The second will remove the given variable if already defined, and the third will set the variable to the literal value given by *value*. Since version 2.0.51 Apache will recognize occurrences of $\$1..\9 within *value* and replace them by parenthesized subexpressions of *regex*.

Example:

```
SetEnvIf Request_URI "\.gif$" object_is_image=gif
SetEnvIf Request_URI "\.jpg$" object_is_image=jpg
SetEnvIf Request_URI "\.xbm$" object_is_image=xbm
:
SetEnvIf Referer www\.\mydomain\.\example\.\com intra_site_referral
:
SetEnvIf object_is_image xbm XBIT_PROCESSING=1
:
SetEnvIf ^TS* ^[a-z].* HAVE_TS
```

The first three will set the environment variable `object_is_image` if the request was for an image file, and the fourth sets `intra_site_referral` if the referring page was somewhere on the `www.mydomain.example.com` Web site.

The last example will set environment variable `HAVE_TS` if the request contains any headers that begin with "TS" whose values begins with any character in the set `[a-z]`.

See also

- [Environment Variables in Apache](#), for additional examples.



Description:	Sets environment variables based on attributes of the request without respect to case
Syntax:	<code>SetEnvIfNoCase <i>attribute regex</i> [!]env-variable[=<i>value</i>] [[!]env-variable[=<i>value</i>]]</code> ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_setenvif

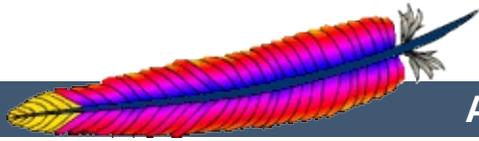
The `SetEnvIfNoCase` is semantically identical to the `SetEnvIf` directive, and differs only in that the regular expression matching is performed in a case-insensitive manner. For example:

```
SetEnvIfNoCase Host Apache\.Org site=apache
```

This will cause the `site` environment variable to be set to "apache" if the HTTP request header field `Host :` was included and contained `Apache.Org`, `apache.org`, or any other combination.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_so

Description:	Loading of executable code and modules into the server at start-up or restart time
Status:	Extension
Module Identifier:	so_module
Source File:	mod_so.c
Compatibility:	This is a Base module (always included) on Windows

Summary

On selected operating systems this module can be used to load modules into Apache at runtime via the [Dynamic Shared Object](#) (DSO) mechanism, rather than requiring a recompilation.

On Unix, the loaded code typically comes from shared object files (usually with .so extension), on Windows this may either be the .so or .dll extension.

Warning

Apache 1.3 modules cannot be directly used with Apache 2.0 - the module must be modified to dynamically load or compile into Apache 2.0.



Note

The module name format changed for Windows with Apache 1.3.15 and 2.0 - the modules are now named as `mod_foo.so`

While `mod_so` still loads modules with `ApacheModuleFoo.dll` names, the new naming convention is preferred; if you are converting your loadable module for 2.0, please fix the name to this 2.0 convention.

The Apache module API is unchanged between the Unix and Windows versions. Many modules will run on Windows with no or little change from Unix, although others rely on aspects of the Unix architecture which are not present in Windows, and will not work.

When a module does work, it can be added to the server in one of two ways. As with Unix, it can be compiled into the server. Because Apache for Windows does not have the `Configure` program of Apache for Unix, the module's source file must be added to the `ApacheCore` project file, and its symbols must be added to the `os\win32\modules.c` file.

The second way is to compile the module as a DLL, a shared library that can be loaded into the server at runtime, using the `LoadModule` directive. These module DLLs can be distributed and run on any Apache for Windows installation, without recompilation of the server.

To create a module DLL, a small change is necessary to the module's source file: The module record must be exported from the DLL (which will be created later; see below). To do this, add the `AP_MODULE_DECLARE_DATA` (defined in the Apache header files) to your module's module record definition. For example, if your module has:

```
module foo_module;
```

Replace the above with:

```
module AP_MODULE_DECLARE_DATA foo_module;
```

Note that this will only be activated on Windows, so the module can continue to be used, unchanged, with Unix if needed. Also, if you are familiar with .DEF files, you can export the module record with that method instead.

Now, create a DLL containing your module. You will need to link this against the libhttpd.lib export library that is created when the libhttpd.dll shared library is compiled. You may also have to change the compiler settings to ensure that the Apache header files are correctly located. You can find this library in your server root's modules directory. It is best to grab an existing module .dsp file from the tree to assure the build environment is configured correctly, or alternately compare the compiler and link options to your .dsp.

This should create a DLL version of your module. Now simply place it in the modules directory of your server root, and use the `LoadModule` directive to load it.



Description:	Link in the named object file or library
Syntax:	LoadFile <i>filename</i> [<i>filename</i>] ...
Context:	server config
Status:	Extension
Module:	mod_so

The LoadFile directive links in the named object files or libraries when the server is started or restarted; this is used to load additional code which may be required for some module to work. *Filename* is either an absolute path or relative to [ServerRoot](#).

For example:

```
LoadFile libexec/libxmlparse.so
```



Description:	Links in the object file or library, and adds to the list of active modules
Syntax:	<code>LoadModule <i>module filename</i></code>
Context:	server config
Status:	Extension
Module:	mod_so

The `LoadModule` directive links in the object file or library *filename* and adds the module structure named *module* to the list of active modules. *Module* is the name of the external variable of type `module` in the file, and is listed as the [Module Identifier](#) in the module documentation.

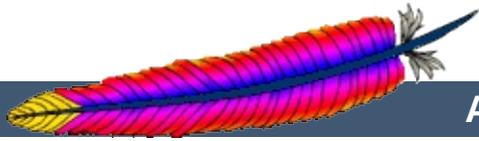
Example:

```
LoadModule status_module modules/mod_status.so
```

loads the named module from the `modules` subdirectory of the `ServerRoot`.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_speling

Description:	Attempts to correct mistaken URLs that users might have entered by ignoring capitalization and by allowing up to one misspelling
Status:	Extension
Module Identifier:	speling_module
Source File:	mod_speling.c

Summary

Requests to documents sometimes cannot be served by the core apache server because the request was misspelled or miscapitalized. This module addresses this problem by trying to find a matching document, even after all other modules gave up. It does its work by comparing each document name in the requested directory against the requested document name **without regard to case**, and allowing **up to one misspelling** (character insertion / omission / transposition or wrong character). A list is built with all document names which were matched using this strategy.

If, after scanning the directory,

- no matching document was found, Apache will proceed as usual and return a "document not found" error.
- only one document is found that "almost" matches the request, then it is returned in the form of a redirection response.
- more than one document with a close match was found, then the list of the matches is returned to the client, and the client can select the correct candidate.



Description:	Limits the action of the spelling module to case corrections
Syntax:	CheckCaseOnly on off
Default:	CheckCaseOnly off
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Extension
Module:	mod_speling

When set, this directive limits the action of the spelling correction to lower/upper case changes. Other potential corrections are not performed.



CheckSpelling Directive

Description:	Enables the spelling module
Syntax:	CheckSpelling on off
Default:	CheckSpelling Off
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Extension
Module:	mod_speling
Compatibility:	CheckSpelling was available as a separately available module for Apache 1.1, but was limited to miscapitalizations. As of Apache 1.3, it is part of the Apache distribution. Prior to Apache 1.3.2, the CheckSpelling directive was only available in the "server" and "virtual host" contexts.

This directive enables or disables the spelling module. When enabled, keep in mind that

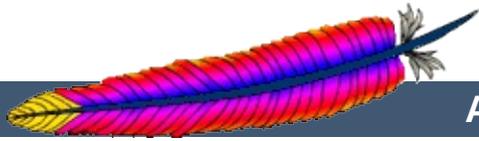
- the directory scan which is necessary for the spelling correction will have an impact on the server's performance when many spelling corrections have to be performed at the same time.
- the document trees should not contain sensitive files which could be matched inadvertently by a spelling "correction".
- the module is unable to correct misspelled user names (as in `http://my.host/~apahce/`), just file names or directory names.
- spelling corrections apply strictly to existing files, so a request for the `<Location /status>` may get incorrectly treated as the negotiated file `"/stats.html"`.

`mod_speling` should not be enabled in [DAV](#) enabled directories, because it will try to "spell fix" newly created resource names against existing filenames, e.g., when trying to upload a new document `doc43.html` it might redirect to an existing document `doc34.html`, which is not what

was intended.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_ssl

Description:	Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols
Status:	Extension
Module Identifier:	ssl_module
Source File:	mod_ssl.c

Summary

This module provides SSL v2/v3 and TLS v1 support for the Apache HTTP Server. It was contributed by Ralf S. Engeschall based on his mod_ssl project and originally derived from work by Ben Laurie.

This module relies on [OpenSSL](#) to provide the cryptography engine.

Further details, discussion, and examples are provided in the [SSL documentation](#).



This module can be configured to provide several items of SSL information as additional environment variables to the SSI and CGI namespace. This information is not provided by default for performance reasons. (See [SSLOptions](#) StdEnvVars, below.) The generated variables are listed in the table below. For backward compatibility the information can be made available under different names, too. Look in the [Compatibility](#) chapter for details on the compatibility variables.

Variable Name:	Value Type:	Description:
HTTPS	flag	HTTPS is being used.
SSL_PROTOCOL	string	The SSL protocol version (SSLv2, SSLv3, TLSv1, TLSv1.1, TLSv1.2)
SSL_SESSION_ID	string	The hex-encoded SSL session id
SSL_CIPHER	string	The cipher specification name
SSL_CIPHER_EXPORT	string	true if cipher is an export cipher
SSL_CIPHER_USEKEYSIZE	number	Number of cipher bits (actually used)
SSL_CIPHER_ALGKEYSIZE	number	Number of cipher bits (possible)
SSL_COMPRESS_METHOD	string	SSL compression method negotiated
SSL_VERSION_INTERFACE	string	The mod_ssl program version
SSL_VERSION_LIBRARY	string	The OpenSSL program version
SSL_CLIENT_M_VERSION	string	The version of the client certificate
SSL_CLIENT_M_SERIAL	string	The serial of the client

		certificate
SSL_CLIENT_S_DN	string	Subject DN in client's certificate
SSL_CLIENT_S_DN_x509	string	Component of client's Subject DN
SSL_CLIENT_I_DN	string	Issuer DN of client's certificate
SSL_CLIENT_I_DN_x509	string	Component of client's Issuer DN
SSL_CLIENT_V_START	string	Validity of client's certificate (start time)
SSL_CLIENT_V_END	string	Validity of client's certificate (end time)
SSL_CLIENT_V_REMAIN	string	Number of days until client's certificate expires
SSL_CLIENT_A_SIG	string	Algorithm used for the signature of client's certificate
SSL_CLIENT_A_KEY	string	Algorithm used for the public key of client's certificate
SSL_CLIENT_CERT	string	PEM-encoded client certificate
SSL_CLIENT_CERT_CHAIN_ <i>n</i>	string	PEM-encoded certificates in client certificate chain
SSL_CLIENT_VERIFY	string	NONE, SUCCESS, GENEROUS or FAILED: <i>reason</i>
SSL_SERVER_M_VERSION	string	The version of the server certificate
SSL_SERVER_M_SERIAL	string	The serial of the server certificate
SSL_SERVER_S_DN	string	Subject DN in server's certificate
SSL_SERVER_S_DN_x509	string	Component of server's Subject

		DN
SSL_SERVER_I_DN	string	Issuer DN of server's certificate
SSL_SERVER_I_DN_x509	string	Component of server's Issuer DN
SSL_SERVER_V_START	string	Validity of server's certificate (start time)
SSL_SERVER_V_END	string	Validity of server's certificate (end time)
SSL_SERVER_A_SIG	string	Algorithm used for the signature of server's certificate
SSL_SERVER_A_KEY	string	Algorithm used for the public key of server's certificate
SSL_SERVER_CERT	string	PEM-encoded server certificate
SSL_TLS_SNI	string	Contents of the SNI TLS extension (if supplied with ClientHello)

x509 specifies a component of an X.509 DN; one of C, ST, L, O, OU, CN, T, I, G, S, D, UID, Email. In Apache 2.1 and later, x509 may also include a numeric _n suffix. If the DN in question contains multiple attributes of the same name, this suffix is used as an index to select a particular attribute. For example, where the server certificate subject DN included two OU fields, SSL_SERVER_S_DN_OU_0 and SSL_SERVER_S_DN_OU_1 could be used to reference each.

SSL_CLIENT_V_REMAIN is only available in version 2.1 and later.



Custom Log Format

When `mod_ssl` is built into Apache or at least loaded (under DSO situation) additional functions exist for the [Custom Log Format](#) of `mod_log_config`. First there is an additional ``%{varname}x` extension format function which can be used to expand any variables provided by any module, especially those provided by `mod_ssl` which can you find in the above table.

For backward compatibility there is additionally a special ``%{name}c` cryptography format function provided. Information about this function is provided in the [Compatibility](#) chapter.

Example

```
CustomLog logs/ssl_request_log \ "%t %h %{SSL_PROTOCOL}x %  
{SSL_CIPHER}x \"%r\" %b"
```

These formats even work without setting the `StdEnvVars` option of the [SSLOptions](#) directive.



Description:	File of concatenated PEM-encoded CA Certificates for Client Auth
Syntax:	<code>SSLCACertificateFile</code> <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificates of Certification Authorities (CA) whose *clients* you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to [SSLCACertificatePath](#).

Example

```
SSLCACertificateFile /usr/local/apache2/conf/ssl.crt/ca-bundle-client.crt
```



Description:	Directory of PEM-encoded CA Certificates for Client Auth
Syntax:	SSLCACertificatePath <i>directory-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificates of Certification Authorities (CAs) whose clients you deal with. These are used to verify the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*.N. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLCACertificatePath /usr/local/apache2/conf/ssl.crt/
```



Description:	File of concatenated PEM-encoded CA Certificates for defining acceptable CA names
Syntax:	<code>SSLCADNRequestFile</code> <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_ssl</code>

When a client certificate is requested by `mod_ssl`, a list of *acceptable Certificate Authority names* is sent to the client in the SSL handshake. These CA names can be used by the client to select an appropriate client certificate out of those it has available.

If neither of the directives `SSLCADNRequestPath` or `SSLCADNRequestFile` are given, then the set of acceptable CA names sent to the client is the names of all the CA certificates given by the `SSLCACertificateFile` and `SSLCACertificatePath` directives; in other words, the names of the CAs which will actually be used to verify the client certificate.

In some circumstances, it is useful to be able to send a set of acceptable CA names which differs from the actual CAs used to verify the client certificate - for example, if the client certificates are signed by intermediate CAs. In such cases, `SSLCADNRequestPath` and/or `SSLCADNRequestFile` can be used; the acceptable CA names are then taken from the complete set of certificates in the directory and/or file specified by this pair of directives.

`SSLCADNRequestFile` must specify an *all-in-one* file containing a concatenation of PEM-encoded CA certificates.

Example

```
SSLCADNRequestFile /usr/local/apache2/conf/ca-names.crt
```



Description:	Directory of PEM-encoded CA Certificates for defining acceptable CA names
Syntax:	SSLCADNRequestPath <i>directory-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This optional directive can be used to specify the set of *acceptable CA names* which will be sent to the client when a client certificate is requested. See the [SSLCADNRequestFile](#) directive for more details.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*.N. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLCADNRequestPath /usr/local/apache2/conf/ca-names.crt/
```



Description:	File of concatenated PEM-encoded CA CRLs for Client Auth
Syntax:	SSLCARevocationFile <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose *clients* you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded CRL files, in order of preference. This can be used alternatively and/or additionally to [SSLCARevocationPath](#).

Example

```
SSLCARevocationFile /usr/local/apache2/conf/ssl.crl/ca-bundle-client.crl
```



Description:	Directory of PEM-encoded CA CRLs for Client Auth
Syntax:	SSLCARevocationPath <i>directory-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) whose clients you deal with. These are used to revoke the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you have not only to place the CRL files there. Additionally you have to create symbolic links named *hash-value*.rN. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLCARevocationPath /usr/local/apache2/conf/ssl.crl/
```



Description:	File of PEM-encoded Server CA Certificates
Syntax:	<code>SSLCertificateChainFile</code> <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_ssl</code>

This directive sets the optional *all-in-one* file where you can assemble the certificates of Certification Authorities (CA) which form the certificate chain of the server certificate. This starts with the issuing CA certificate of the server certificate and can range up to the root CA certificate. Such a file is simply the concatenation of the various PEM-encoded CA Certificate files, usually in certificate chain order.

This should be used alternatively and/or additionally to [SSLCACertificatePath](#) for explicitly constructing the server certificate chain which is sent to the browser in addition to the server certificate. It is especially useful to avoid conflicts with CA certificates when using client authentication. Because although placing a CA certificate of the server certificate chain into [SSLCACertificatePath](#) has the same effect for the certificate chain construction, it has the side-effect that client certificates issued by this same CA certificate are also accepted on client authentication.

But be careful: Providing the certificate chain works only if you are using a *single* RSA or DSA based server certificate. If you are using a coupled RSA+DSA certificate pair, this will work only if actually both certificates use the *same* certificate chain. Else the browsers will be confused in this situation.

Example

```
SSLCertificateChainFile /usr/local/apache2/conf/ssl.crt/ca.crt
```



Description:	Server PEM-encoded X.509 Certificate file
Syntax:	<code>SSLCertificateFile</code> <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_ssl</code>
Compatibility:	ECC support is available in Apache 2.2.26 and later

This directive points to a file with certificate data in PEM format. At a minimum, the file must include an end-entity (leaf) certificate. The directive can be used up to three times (referencing different filenames) when an RSA, a DSA, and an ECC based server certificate is used in parallel.

Custom DH parameters and an EC curve name for ephemeral keys, can be added to end of the first file configured using [SSLCertificateFile](#). This is supported in version 2.2.30 or later. Such parameters can be generated using the commands `openssl dhparam` and `openssl ecparam`. The parameters can be added as-is to the end of the first certificate file. Only the first file can be used for custom parameters, as they are applied independently of the authentication algorithm type.

Finally the the end-entity certificate's private key can also be added to the certificate file instead of using a separate [SSLCertificateKeyFile](#) directive. This practice is highly discouraged. If the private key is encrypted, the pass phrase dialog is forced at startup time.

DH parameter interoperability with primes > 1024 bit

Beginning with version 2.2.30, `mod_ssl` makes use of standardized DH parameters with prime lengths of 2048, 3072, 4096, 6144 and 8192 bits (from [RFC 3526](#)), and hands them out to clients based on the length of the certificate's RSA/DSA key. With Java-based clients in particular (Java 7 or earlier), this may lead to handshake failures - see

this [FAQ answer](#) for working around such issues.

Example

```
SSLCertificateFile /usr/local/apache2/conf/ssl.crt/server.crt
```



Description:	Server PEM-encoded Private Key file
Syntax:	SSLCertificateKeyFile <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	ECC support is available in Apache 2.2.26 and later

This directive points to the PEM-encoded private key file for the server. If the contained private key is encrypted, the pass phrase dialog is forced at startup time.

The directive can be used up to three times (referencing different filenames) when an RSA, a DSA, and an ECC based private key is used in parallel. For each [SSLCertificateKeyFile](#) directive, there must be a matching [SSLCertificateFile](#) directive.

The private key may also be combined with the certificate in the file given by [SSLCertificateFile](#), but this practice is highly discouraged.

Example

```
SSLCertificateKeyFile /usr/local/apache2/conf/ssl.key/server.key
```



Description:	Cipher Suite available for negotiation in SSL handshake
Syntax:	SSLCipherSuite <i>cipher-spec</i>
Default:	SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:-
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This complex directive uses a colon-separated *cipher-spec* string consisting of OpenSSL cipher specifications to configure the Cipher Suite the client is permitted to negotiate in the SSL handshake phase. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured Cipher Suite after the HTTP request was read but before the HTTP response is sent.

An SSL cipher specification in *cipher-spec* is composed of 4 major attributes plus a few extra minor ones:

- *Key Exchange Algorithm:*
RSA or Diffie-Hellman variants.
- *Authentication Algorithm:*
RSA, Diffie-Hellman, DSS or none.
- *Cipher/Encryption Algorithm:*
DES, Triple-DES, RC4, RC2, IDEA or none.
- *MAC Digest Algorithm:*
MD5, SHA or SHA1.

An SSL cipher can also be an export cipher and is either a SSLv2 or SSLv3/TLSv1 cipher (here TLSv1 is equivalent to SSLv3). To specify which ciphers to use, one can either specify all the Ciphers, one at a

time, or use aliases to specify the preference and order for the ciphers (see [Table 1](#)).

Tag	Description
<i>Key Exchange Algorithm:</i>	
kRSA	RSA key exchange
kDHR	Diffie-Hellman key exchange with RSA key
kDHd	Diffie-Hellman key exchange with DSA key
kEDH	Ephemeral (temp.key) Diffie-Hellman key exchange (no cert)
<i>Authentication Algorithm:</i>	
aNULL	No authentication
aRSA	RSA authentication
aDSS	DSS authentication
aDH	Diffie-Hellman authentication
<i>Cipher Encoding Algorithm:</i>	
eNULL	No encoding
DES	DES encoding
3DES	Triple-DES encoding
RC4	RC4 encoding
RC2	RC2 encoding
IDEA	IDEA encoding
<i>MAC Digest Algorithm:</i>	
MD5	MD5 hash function
SHA1	SHA1 hash function
SHA	SHA hash function
<i>Aliases:</i>	
SSLv2	all SSL version 2.0 ciphers
SSLv3	all SSL version 3.0 ciphers

TLSv1	all TLS version 1.0 ciphers
EXP	all export ciphers
EXPORT40	all 40-bit export ciphers only
EXPORT56	all 56-bit export ciphers only
LOW	all low strength ciphers (no export, single DES)
MEDIUM	all ciphers with 128 bit encryption
HIGH	all ciphers using Triple-DES
RSA	all ciphers using RSA key exchange
DH	all ciphers using Diffie-Hellman key exchange
EDH	all ciphers using Ephemeral Diffie-Hellman key exchange
ADH	all ciphers using Anonymous Diffie-Hellman key exchange
DSS	all ciphers using DSS authentication
NULL	all ciphers using no encryption

Now where this becomes interesting is that these can be put together to specify the order and ciphers you wish to use. To speed this up there are also aliases (SSLv2, SSLv3, TLSv1, EXP, LOW, MEDIUM, HIGH) for certain groups of ciphers. These tags can be joined together with prefixes to form the *cipher-spec*. Available prefixes are:

- none: add cipher to list
- +: move matching ciphers to the current location in list
- -: remove cipher from list (can be added later again)
- !: kill cipher from list completely (can **not** be added later again)

aNULL, eNULL and EXP ciphers are always disabled

Beginning with version 2.2.30, null and export-grade ciphers are always disabled, as `mod_ssl` unconditionally prepends any supplied cipher suite string with `!aNULL:!eNULL:!EXP:` at initialization.

A simpler way to look at all of this is to use the `openssl ciphers -v`

command which provides a nice way to successively create the correct *cipher-spec* string. The default *cipher-spec* string is ``ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP`` which means the following: first, remove from consideration any ciphers that do not authenticate, i.e. for SSL only the Anonymous Diffie-Hellman ciphers. Next, use ciphers using RC4 and RSA. Next include the high, medium and then the low security ciphers. Finally *pull* all SSLv2 and export ciphers to the end of the list.

```
$ openssl ciphers -v 'ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP'
NULL-SHA          SSLv3 Kx=RSA      Au=RSA  Enc=None  Mac=SHA1
NULL-MD5          SSLv3 Kx=RSA      Au=RSA  Enc=None  Mac=MD5
EDH-RSA-DES-CBC3-SHA  SSLv3 Kx=DH       Au=RSA  Enc=3DES(168) Mac=SHA1
...
EXP-RC4-MD5       SSLv3 Kx=RSA(512) Au=RSA  Enc=RC4(40) Mac=MD5
EXP-RC2-CBC-MD5   SSLv2 Kx=RSA(512) Au=RSA  Enc=RC2(40) Mac=MD5
EXP-RC4-MD5       SSLv2 Kx=RSA(512) Au=RSA  Enc=RC4(40) Mac=MD5
```

The complete list of particular RSA & DH ciphers for SSL is given in [Table 2](#).

Example

```
SSLCipherSuite RSA:!EXP:!NULL:+HIGH:+MEDIUM:-LOW
```

Cipher-Tag	Protocol	Key Ex.	Auth.	Enc.	MAC	Type
<i>RSA Ciphers:</i>						
DES-CBC3-SHA	SSLv3	RSA	RSA	3DES(168)	SHA1	
DES-CBC3-MD5	SSLv2	RSA	RSA	3DES(168)	MD5	
IDEA-CBC-SHA	SSLv3	RSA	RSA	IDEA(128)	SHA1	
RC4-SHA	SSLv3	RSA	RSA	RC4(128)	SHA1	
RC4-MD5	SSLv3	RSA	RSA	RC4(128)	MD5	
IDEA-CBC-MD5	SSLv2	RSA	RSA	IDEA(128)	MD5	
RC2-CBC-MD5	SSLv2	RSA	RSA	RC2(128)	MD5	

RC4 - MD5	SSLv2	RSA	RSA	RC4(128)	MD5	
DES - CBC - SHA	SSLv3	RSA	RSA	DES(56)	SHA1	
RC4 - 64 - MD5	SSLv2	RSA	RSA	RC4(64)	MD5	
DES - CBC - MD5	SSLv2	RSA	RSA	DES(56)	MD5	
EXP - DES - CBC - SHA	SSLv3	RSA(512)	RSA	DES(40)	SHA1	export
EXP - RC2 - CBC - MD5	SSLv3	RSA(512)	RSA	RC2(40)	MD5	export
EXP - RC4 - MD5	SSLv3	RSA(512)	RSA	RC4(40)	MD5	export
EXP - RC2 - CBC - MD5	SSLv2	RSA(512)	RSA	RC2(40)	MD5	export
EXP - RC4 - MD5	SSLv2	RSA(512)	RSA	RC4(40)	MD5	export
NULL - SHA	SSLv3	RSA	RSA	None	SHA1	
NULL - MD5	SSLv3	RSA	RSA	None	MD5	
<i>Diffie-Hellman Ciphers:</i>						
ADH - DES - CBC3 - SHA	SSLv3	DH	None	3DES(168)	SHA1	
ADH - DES - CBC - SHA	SSLv3	DH	None	DES(56)	SHA1	
ADH - RC4 - MD5	SSLv3	DH	None	RC4(128)	MD5	
EDH - RSA - DES - CBC3 - SHA	SSLv3	DH	RSA	3DES(168)	SHA1	
EDH - DSS - DES - CBC3 - SHA	SSLv3	DH	DSS	3DES(168)	SHA1	
EDH - RSA - DES - CBC - SHA	SSLv3	DH	RSA	DES(56)	SHA1	
EDH - DSS - DES - CBC - SHA	SSLv3	DH	DSS	DES(56)	SHA1	
EXP - EDH - RSA - DES - CBC - SHA	SSLv3	DH(512)	RSA	DES(40)	SHA1	export

EXP-EDH-DSS-DES-CBC-SHA	SSLv3	DH(512)	DSS	DES(40)	SHA1	export
EXP-ADH-DES-CBC-SHA	SSLv3	DH(512)	None	DES(40)	SHA1	export
EXP-ADH-RC4-MD5	SSLv3	DH(512)	None	RC4(40)	MD5	export



Description:	Enable compression on the SSL level
Syntax:	SSLCompression on off
Default:	SSLCompression off
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.2.24 and later, if using OpenSSL 0.9.8 or later; virtual host scope available if using OpenSSL 1.0.0 or later. The default used to be on in versions 2.2.24 to 2.2.25.

This directive allows to enable compression on the SSL level.

Enabling compression causes security issues in most setups (the so called CRIME attack).



Description:	Enable use of a cryptographic hardware accelerator
Syntax:	SSLCryptoDevice <i>engine</i>
Default:	SSLCryptoDevice builtin
Context:	server config
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in Apache 2.1 and later, if using -engine flavor of OpenSSL 0.9.6, or OpenSSL 0.9.7 or later

This directive enables use of a cryptographic hardware accelerator board to offload some of the SSL processing overhead. This directive can only be used if the SSL toolkit is built with "engine" support; OpenSSL 0.9.7 and later releases have "engine" support by default, the separate "-engine" releases of OpenSSL 0.9.6 must be used.

To discover which engine names are supported, run the command "openssl engine".

Example

```
# For a Broadcom accelerator:  
SSLCryptoDevice ubsec
```



Description:	SSL Engine Operation Switch
Syntax:	SSLEngine on off optional
Default:	SSLEngine off
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive toggles the usage of the SSL/TLS Protocol Engine. This should be used inside a `<VirtualHost>` section to enable SSL/TLS for a that virtual host. By default the SSL/TLS Protocol Engine is disabled for both the main server and all configured virtual hosts.

Example

```
<VirtualHost _default_:443>  
SSLEngine on  
...  
</VirtualHost>
```

In Apache 2.1 and later, `SSLEngine` can be set to `optional`. This enables support for [RFC 2817](#), Upgrading to TLS Within HTTP/1.1. At this time no web browsers support RFC 2817.



Description:	SSL FIPS mode Switch
Syntax:	SSLFIPS on off
Default:	SSLFIPS off
Context:	server config
Status:	Extension
Module:	mod_ssl

This directive toggles the usage of the SSL library FIPS_mode flag. It must be set in the global server context and cannot be configured with conflicting settings (SSLFIPS on followed by SSLFIPS off or similar). The mode applies to all SSL library operations.

If httpd was compiled against an SSL library which did not support the FIPS_mode flag, SSLFIPS on will fail. Refer to the FIPS 140-2 Security Policy document of the SSL provider library for specific requirements to use mod_ssl in a FIPS 140-2 approved mode of operation; note that mod_ssl itself is not validated, but may be described as using FIPS 140-2 validated cryptographic module, when all components are assembled and operated under the guidelines imposed by the applicable Security Policy.



Description:	Option to prefer the server's cipher preference order
Syntax:	SSLHonorCipherOrder <i>flag</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in Apache 2.1 and later, if using OpenSSL 0.9.7 or later

When choosing a cipher during an SSLv3 or TLSv1 handshake, normally the client's preference is used. If this directive is enabled, the server's preference will be used instead.

Example

```
SSLHonorCipherOrder on
```



Description:	Option to enable support for insecure renegotiation
Syntax:	SSLInsecureRenegotiation <i>flag</i>
Default:	SSLInsecureRenegotiation off
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.2.15 and later, if using OpenSSL 0.9.8m or later

As originally specified, all versions of the SSL and TLS protocols (up to and including TLS/1.2) were vulnerable to a Man-in-the-Middle attack ([CVE-2009-3555](#)) during a renegotiation. This vulnerability allowed an attacker to "prefix" a chosen plaintext to the HTTP request as seen by the web server. A protocol extension was developed which fixed this vulnerability if supported by both client and server.

If `mod_ssl` is linked against OpenSSL version 0.9.8m or later, by default renegotiation is only supported with clients supporting the new protocol extension. If this directive is enabled, renegotiation will be allowed with old (unpatched) clients, albeit insecurely.

Security warning

If this directive is enabled, SSL connections will be vulnerable to the Man-in-the-Middle prefix attack as described in [CVE-2009-3555](#).

Example

```
SSLInsecureRenegotiation on
```

The `SSL_SECURE_RENEG` environment variable can be used from an SSI or CGI script to determine whether secure renegotiation is supported for a given SSL connection.



Description:	Semaphore for internal mutual exclusion of operations
Syntax:	SSLMutex <i>type</i>
Default:	SSLMutex none
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures the SSL engine's semaphore (aka. lock) which is used for mutual exclusion of operations which have to be done in a synchronized way between the pre-forked Apache server processes. This directive can only be used in the global server context because it's only useful to have one global mutex. This directive is designed to closely match the [AcceptMutex](#) directive.

The following Mutex *types* are available:

- none | no
This is the default where no Mutex is used at all. Use it at your own risk. But because currently the Mutex is mainly used for synchronizing write access to the SSL Session Cache you can live without it as long as you accept a sometimes garbled Session Cache. So it's not recommended to leave this the default. Instead configure a real Mutex.
- posixsem
This is an elegant Mutex variant where a Posix Semaphore is used when possible. It is only available when the underlying platform and [APR](#) supports it.
- sysvsem
This is a somewhat elegant Mutex variant where a SystemV IPC Semaphore is used when possible. It is possible to "leak" SysV semaphores if processes crash before the semaphore is removed. It

is only available when the underlying platform and [APR](#) supports it.

- `sem`

This directive tells the SSL Module to pick the "best" semaphore implementation available to it, choosing between Posix and SystemV IPC, in that order. It is only available when the underlying platform and [APR](#) supports at least one of the 2.

- `pthread`

This directive tells the SSL Module to use Posix thread mutexes. It is only available if the underlying platform and [APR](#) supports it.

- `fcntl:/path/to/mutex`

This is a portable Mutex variant where a physical (lock-)file and the `fcntl()` function are used as the Mutex. Always use a local disk filesystem for `/path/to/mutex` and never a file residing on a NFS- or AFS-filesystem. It is only available when the underlying platform and [APR](#) supports it. Note: Internally, the Process ID (PID) of the Apache parent process is automatically appended to `/path/to/mutex` to make it unique, so you don't have to worry about conflicts yourself. Notice that this type of mutex is not available under the Win32 environment. There you *have* to use the semaphore mutex.

- `flock:/path/to/mutex`

This is similar to the `fcntl:/path/to/mutex` method with the exception that the `flock()` function is used to provide file locking. It is only available when the underlying platform and [APR](#) supports it.

- `file:/path/to/mutex`

This directive tells the SSL Module to pick the "best" file locking implementation available to it, choosing between `fcntl` and `flock`, in that order. It is only available when the underlying platform and [APR](#) supports at least one of the 2.

- default | yes

This directive tells the SSL Module to pick the default locking implementation as determined by the platform and [APR](#).

Example

```
SSLMutex file:/usr/local/apache/logs/ssl_mutex
```



Description:	Configure various SSL engine run-time options
Syntax:	SSLOptions [+ -] <i>option</i> ...
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Extension
Module:	mod_ssl

This directive can be used to control various run-time options on a per-directory basis. Normally, if multiple `SSLOptions` could apply to a directory, then the most specific one is taken completely; the options are not merged. However if *all* the options on the `SSLOptions` directive are preceded by a plus (+) or minus (-) symbol, the options are merged. Any options preceded by a + are added to the options currently in force, and any options preceded by a - are removed from the options currently in force.

The available *options* are:

- `StdEnvVars`

When this option is enabled, the standard set of SSL related CGI/SSI environment variables are created. This per default is disabled for performance reasons, because the information extraction step is a rather expensive operation. So one usually enables this option for CGI and SSI requests only.

- `ExportCertData`

When this option is enabled, additional CGI/SSI environment variables are created: `SSL_SERVER_CERT`, `SSL_CLIENT_CERT` and `SSL_CLIENT_CERT_CHAIN_n` (with $n = 0,1,2,..$). These contain the PEM-encoded X.509 Certificates of server and client for the current HTTPS connection and can be used by CGI scripts for deeper Certificate checking. Additionally all other certificates of the client

certificate chain are provided, too. This bloats up the environment a little bit which is why you have to use this option to enable it on demand.

- `FakeBasicAuth`

When this option is enabled, the Subject Distinguished Name (DN) of the Client X509 Certificate is translated into a HTTP Basic Authorization username. This means that the standard Apache authentication methods can be used for access control. The user name is just the Subject of the Client's X509 Certificate (can be determined by running OpenSSL's `openssl x509` command: `openssl x509 -noout -subject -in certificate.crt`). Note that no password is obtained from the user. Every entry in the user file needs this password: ``xj31ZMTZzkVA"`, which is the DES-encrypted version of the word ``password"`. Those who live under MD5-based encryption (for instance under FreeBSD or BSD/OS, etc.) should use the following MD5 hash of the same word: ``$1$0XLYS...$0wx8s2/m9/gfkcRVXzgoE/".`

- `StrictRequire`

This *forces* forbidden access when `SSLRequireSSL` or `SSLRequire` successfully decided that access should be forbidden. Usually the default is that in the case where a ``Satisfy any"` directive is used, and other access restrictions are passed, denial of access due to `SSLRequireSSL` or `SSLRequire` is overridden (because that's how the Apache `Satisfy` mechanism should work.) But for strict access restriction you can use `SSLRequireSSL` and/or `SSLRequire` in combination with an ``SSLOptions +StrictRequire"`. Then an additional ``Satisfy Any"` has no chance once `mod_ssl` has decided to deny access.

- `OptRenegotiate`

This enables optimized SSL connection renegotiation handling when SSL directives are used in per-directory context. By default a strict

scheme is enabled where *every* per-directory reconfiguration of SSL parameters causes a *full* SSL renegotiation handshake. When this option is used `mod_ssl` tries to avoid unnecessary handshakes by doing more granular (but still safe) parameter checks. Nevertheless these granular checks sometimes maybe not what the user expects, so enable this on a per-directory basis only, please.

Example

```
SSLOptions +FakeBasicAuth -StrictRequire  
<Files ~ "\.(cgi|html)$">  
SSLOptions +StdEnvVars -ExportCertData  
<Files>
```



Description:	Type of pass phrase dialog for encrypted private keys
Syntax:	SSLPassPhraseDialog <i>type</i>
Default:	SSLPassPhraseDialog builtin
Context:	server config
Status:	Extension
Module:	mod_ssl

When Apache starts up it has to read the various Certificate (see [SSLCertificateFile](#)) and Private Key (see [SSLCertificateKeyFile](#)) files of the SSL-enabled virtual servers. Because for security reasons the Private Key files are usually encrypted, mod_ssl needs to query the administrator for a Pass Phrase in order to decrypt those files. This query can be done in two ways which can be configured by *type*:

- **builtin**

This is the default where an interactive terminal dialog occurs at startup time just before Apache detaches from the terminal. Here the administrator has to manually enter the Pass Phrase for each encrypted Private Key file. Because a lot of SSL-enabled virtual hosts can be configured, the following reuse-scheme is used to minimize the dialog: When a Private Key file is encrypted, all known Pass Phrases (at the beginning there are none, of course) are tried. If one of those known Pass Phrases succeeds no dialog pops up for this particular Private Key file. If none succeeded, another Pass Phrase is queried on the terminal and remembered for the next round (where it perhaps can be reused).

This scheme allows mod_ssl to be maximally flexible (because for N encrypted Private Key files you *can* use N different Pass Phrases - but then you have to enter all of them, of course) while minimizing the terminal dialog (i.e. when you use a single Pass Phrase for all N Private Key files this Pass Phrase is queried only once).

- `|/path/to/program [args...]`

This mode allows an external program to be used which acts as a pipe to a particular input device; the program is sent the standard prompt text used for the `builtin` mode on `stdin`, and is expected to write password strings on `stdout`. If several passwords are needed (or an incorrect password is entered), additional prompt text will be written subsequent to the first password being returned, and more passwords must then be written back.

- `exec:/path/to/program`

Here an external program is configured which is called at startup for each encrypted Private Key file. It is called with two arguments (the first is of the form `servername:portnumber`, the second is either `RSA`, `DSA`, or `ECC`), which indicate for which server and algorithm it has to print the corresponding Pass Phrase to `stdout`. The intent is that this external program first runs security checks to make sure that the system is not compromised by an attacker, and only when these checks were passed successfully it provides the Pass Phrase.

Both these security checks, and the way the Pass Phrase is determined, can be as complex as you like. `Mod_ssl` just defines the interface: an executable program which provides the Pass Phrase on `stdout`. Nothing more or less! So, if you're really paranoid about security, here is your interface. Anything else has to be left as an exercise to the administrator, because local security requirements are so different.

The `reuse-algorithm` above is used here, too. In other words: The external program is called only once per unique Pass Phrase.

Example

```
SSLPassPhraseDialog exec:/usr/local/apache/sbin/pp-filter
```



Description:	Configure usable SSL protocol flavors
Syntax:	SSLProtocol [+ -] <i>protocol</i> ...
Default:	SSLProtocol all
Context:	server config, virtual host
Override:	Options
Status:	Extension
Module:	mod_ssl

This directive can be used to control the SSL protocol flavors `mod_ssl` should use when establishing its server environment. Clients then can only connect with one of the provided protocols.

The available (case-insensitive) *protocols* are:

- **SSLv2**
This is the Secure Sockets Layer (SSL) protocol, version 2.0. It is the original SSL protocol as designed by Netscape Corporation. Though its use has been deprecated, because of weaknesses in the security of the protocol.
- **SSLv3**
This is the Secure Sockets Layer (SSL) protocol, version 3.0, from the Netscape Corporation. It is the successor to SSLv2 and the predecessor to TLSv1. It's supported by almost all popular browsers.
- **TLSv1**
This is the Transport Layer Security (TLS) protocol, version 1.0. It is the successor to SSLv3 and is defined in [RFC 2246](#).
- **TLSv1.1** (when using OpenSSL 1.0.1 and later)
A revision of the TLS 1.0 protocol, as defined in [RFC 4346](#).
- **TLSv1.2** (when using OpenSSL 1.0.1 and later)

A revision of the TLS 1.1 protocol, as defined in [RFC 5246](#).

- All

This is a shortcut for ``+SSLv2 +SSLv3 +TLSv1" or - when using OpenSSL 1.0.1 and later - ``+SSLv2 +SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2", respectively.

Example

```
# enable SSLv3 and all available TLSv1 flavors, but not SSLv2
SSLProtocol All -SSLv2
```



Description:	File of concatenated PEM-encoded CA Certificates for Remote Server Auth
Syntax:	SSLProxyCACertificateFile <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificates of Certification Authorities (CA) whose *remote servers* you deal with. These are used for Remote Server Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to [SSLProxyCACertificatePath](#).

Example

```
SSLProxyCACertificateFile /usr/local/apache2/conf/ssl.crt/ca-bundle-remote-server.crt
```



Description:	Directory of PEM-encoded CA Certificates for Remote Server Auth
Syntax:	SSLProxyCACertificatePath <i>directory-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificates of Certification Authorities (CAs) whose remote servers you deal with. These are used to verify the remote server certificate on Remote Server Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*.N. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLProxyCACertificatePath /usr/local/apache2/conf/ssl.crt/
```



Description:	File of concatenated PEM-encoded CA CRLs for Remote Server Auth
Syntax:	SSLProxyCARevocationFile <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose *remote servers* you deal with. These are used for Remote Server Authentication. Such a file is simply the concatenation of the various PEM-encoded CRL files, in order of preference. This can be used alternatively and/or additionally to [SSLProxyCARevocationPath](#).

Example

```
SSLProxyCARevocationFile /usr/local/apache2/conf/ssl.crl/ca-bundle-remote-server.crl
```



Description:	Directory of PEM-encoded CA CRLs for Remote Server Auth
Syntax:	SSLProxyCAREvocationPath <i>directory-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) whose remote servers you deal with. These are used to revoke the remote server certificate on Remote Server Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you have not only to place the CRL files there. Additionally you have to create symbolic links named *hash-value*.rN. And you should always make sure this directory contains the appropriate symbolic links.

Example

```
SSLProxyCAREvocationPath /usr/local/apache2/conf/ssl.crl/
```



Description:	Whether to check the remote server certificates CN field
Syntax:	SSLProxyCheckPeerCN on off
Default:	SSLProxyCheckPeerCN off
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets whether the remote server certificates CN field is compared against the hostname of the request URL. If both are not equal a 502 status code (Bad Gateway) is sent.

Example

```
SSLProxyCheckPeerCN on
```



Description:	Whether to check if remote server certificate is expired
Syntax:	SSLProxyCheckPeerExpire on off
Default:	SSLProxyCheckPeerExpire off
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets whether it is checked if the remote server certificate is expired or not. If the check fails a 502 status code (Bad Gateway) is sent.

Example

```
SSLProxyCheckPeerExpire on
```



Description:	Cipher Suite available for negotiation in SSL proxy handshake
Syntax:	SSLProxyCipherSuite <i>cipher-spec</i>
Default:	SSLProxyCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:-
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

Equivalent to `SSLCipherSuite`, but for the proxy connection. Please refer to [SSLCipherSuite](#) for additional information.



Description:	SSL Proxy Engine Operation Switch
Syntax:	SSLProxyEngine on off
Default:	SSLProxyEngine off
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive toggles the usage of the SSL/TLS Protocol Engine for proxy. This is usually used inside a [<VirtualHost>](#) section to enable SSL/TLS for proxy usage in a particular virtual host. By default the SSL/TLS Protocol Engine is disabled for proxy both for the main server and all configured virtual hosts.

Note that the SSLProxyEngine directive should not, in general, be included in a virtual host that will be acting as a forward proxy (using [<Proxy>](#) or [<ProxyRequest>](#) directives). SSLProxyEngine is not required to enable a forward proxy server to proxy SSL/TLS requests.

Example

```
<VirtualHost _default_:443>  
SSLProxyEngine on  
...  
</VirtualHost>
```



Description:	File of concatenated PEM-encoded CA certificates to be used by the proxy for choosing a certificate
Syntax:	SSLProxyMachineCertificateChainFile <i>filename</i>
Context:	server config
Override:	Not applicable
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in Apache 2.2.23 and later

This directive sets the all-in-one file where you keep the certificate chain for all of the client certs in use. This directive will be needed if the remote server presents a list of CA certificates that are not direct signers of one of the configured client certificates.

This referenced file is simply the concatenation of the various PEM-encoded certificate files. Upon startup, each client certificate configured will be examined and a chain of trust will be constructed.

Security warning

If this directive is enabled, all of the certificates in the file will be trusted as if they were also in [SSLProxyCACertificateFile](#).

Example

```
SSLProxyMachineCertificateChainFile  
/usr/local/apache2/conf/ssl.crt/proxyCA.pem
```



Description:	File of concatenated PEM-encoded client certificates and keys to be used by the proxy
Syntax:	SSLProxyMachineCertificateFile <i>filename</i>
Context:	server config
Override:	Not applicable
Status:	Extension
Module:	mod_ssl

This directive sets the all-in-one file where you keep the certificates and keys used for authentication of the proxy server to remote servers.

This referenced file is simply the concatenation of the various PEM-encoded certificate files, in order of preference. Use this directive alternatively or additionally to `SSLProxyMachineCertificatePath`.

Currently there is no support for encrypted private keys

Example

```
SSLProxyMachineCertificateFile  
/usr/local/apache2/conf/ssl.crt/proxy.pem
```



Description:	Directory of PEM-encoded client certificates and keys to be used by the proxy
Syntax:	SSLProxyMachineCertificatePath <i>directory</i>
Context:	server config
Override:	Not applicable
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the certificates and keys used for authentication of the proxy server to remote servers.

The files in this directory must be PEM-encoded and are accessed through hash filenames. Additionally, you must create symbolic links named *hash-value*.N. And you should always make sure this directory contains the appropriate symbolic links.

Currently there is no support for encrypted private keys

Example

```
SSLProxyMachineCertificatePath /usr/local/apache2/conf/proxy.crt/
```



Description:	Configure usable SSL protocol flavors for proxy usage
Syntax:	SSLProxyProtocol [+ -] <i>protocol</i> ...
Default:	SSLProxyProtocol all
Context:	server config, virtual host
Override:	Options
Status:	Extension
Module:	mod_ssl

This directive can be used to control the SSL protocol flavors mod_ssl should use when establishing its server environment for proxy . It will only connect to servers using one of the provided protocols.

Please refer to [SSLProtocol](#) for additional information.



Description:	Type of remote server Certificate verification
Syntax:	SSLProxyVerify <i>level</i>
Default:	SSLProxyVerify none
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

When a proxy is configured to forward requests to a remote SSL server, this directive can be used to configure certificate verification of the remote server.

Note that even when certificate verification is enabled, `mod_ssl` does **not** check whether the `commonName` (hostname) attribute of the server certificate matches the hostname used to connect to the server. In other words, the proxy does not guarantee that the SSL connection to the backend server is "secure" beyond the fact that the certificate is signed by one of the CAs configured using the `SSLProxyCACertificatePath` and/or `SSLProxyCACertificateFile` directives. In order to get this check done please have a look at `SSLProxyCheckPeerCN` and `SSLProxyCheckPeerExpire` directives which are off by default.

The following levels are available for *level*:

- **none**: no remote server Certificate is required at all
- **optional**: the remote server *may* present a valid Certificate
- **require**: the remote server *has to* present a valid Certificate
- **optional_no_ca**: the remote server may present a valid Certificate but it need not to be (successfully) verifiable.

In practice only levels **none** and **require** are really interesting, because level **optional** doesn't work with all servers and level **optional_no_ca** is

actually against the idea of authentication (but can be used to establish SSL test pages, etc.)

Example

```
SSLProxyVerify require
```



Description:	Maximum depth of CA Certificates in Remote Server Certificate verification
Syntax:	SSLProxyVerifyDepth <i>number</i>
Default:	SSLProxyVerifyDepth 1
Context:	server config, virtual host
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets how deeply mod_ssl should verify before deciding that the remote server does not have a valid certificate.

The depth actually is the maximum number of intermediate certificate issuers, i.e. the number of CA certificates which are max allowed to be followed while verifying the remote server certificate. A depth of 0 means that self-signed remote server certificates are accepted only, the default depth of 1 means the remote server certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA's certificate is under [SSLProxyCACertificatePath](#)), etc.

Example

```
SSLProxyVerifyDepth 10
```



Description:	Pseudo Random Number Generator (PRNG) seeding source
Syntax:	SSLRandomSeed <i>context source [bytes]</i>
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures one or more sources for seeding the Pseudo Random Number Generator (PRNG) in OpenSSL at startup time (*context* is `start up`) and/or just before a new SSL connection is established (*context* is `connect`). This directive can only be used in the global server context because the PRNG is a global facility.

The following *source* variants are available:

- `builtin`
This is the always available builtin seeding source. Its usage consumes minimum CPU cycles under runtime and hence can be always used without drawbacks. The source used for seeding the PRNG contains of the current time, the current process id and (when applicable) a randomly chosen 1KB extract of the inter-process scoreboard structure of Apache. The drawback is that this is not really a strong source and at startup time (where the scoreboard is still not available) this source just produces a few bytes of entropy. So you should always, at least for the startup, use an additional seeding source.
- `file:/path/to/source`
This variant uses an external file `/path/to/source` as the source for seeding the PRNG. When *bytes* is specified, only the first *bytes* number of bytes of the file form the entropy (and *bytes* is given to `/path/to/source` as the first argument). When *bytes* is not specified the whole file forms the entropy (and `0` is given to

`/path/to/source` as the first argument). Use this especially at startup time, for instance with an available `/dev/random` and/or `/dev/urandom` devices (which usually exist on modern Unix derivatives like FreeBSD and Linux).

But be careful: Usually `/dev/random` provides only as much entropy data as it actually has, i.e. when you request 512 bytes of entropy, but the device currently has only 100 bytes available two things can happen: On some platforms you receive only the 100 bytes while on other platforms the read blocks until enough bytes are available (which can take a long time). Here using an existing `/dev/urandom` is better, because it never blocks and actually gives the amount of requested data. The drawback is just that the quality of the received data may not be the best.

On some platforms like FreeBSD one can even control how the entropy is actually generated, i.e. by which system interrupts. More details one can find under *rndcontrol(8)* on those platforms.

Alternatively, when your system lacks such a random device, you can use a tool like [EGD](#) (Entropy Gathering Daemon) and run its client program with the `exec:/path/to/program/` variant (see below) or use `egd:/path/to/egd-socket` (see below).

- `exec:/path/to/program`

This variant uses an external executable `/path/to/program` as the source for seeding the PRNG. When *bytes* is specified, only the first *bytes* number of bytes of its `stdout` contents form the entropy. When *bytes* is not specified, the entirety of the data produced on `stdout` form the entropy. Use this only at startup time when you need a very strong seeding with the help of an external program (for instance as in the example above with the `truerand` utility you can find in the `mod_ssl` distribution which is based on the AT&T *truerand* library). Using this in the connection context slows down the server too dramatically, of course. So usually you should avoid using

external programs in that context.

- `egd:/path/to/egd-socket` (Unix only)
This variant uses the Unix domain socket of the external Entropy Gathering Daemon (EGD) (see <http://www.lothar.com/tech/crypto/>) to seed the PRNG. Use this if no random device exists on your platform.

Example

```
SSLRandomSeed startup builtin
SSLRandomSeed startup file:/dev/random
SSLRandomSeed startup file:/dev/urandom 1024
SSLRandomSeed startup exec:/usr/local/bin/truerand 16
SSLRandomSeed connect builtin
SSLRandomSeed connect file:/dev/random
SSLRandomSeed connect file:/dev/urandom 1024
```



Description:	Set the size for the SSL renegotiation buffer
Syntax:	SSLRenegBufferSize <i>bytes</i>
Default:	SSLRenegBufferSize 131072
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

If an SSL renegotiation is required in per-location context, for example, any use of [SSLVerifyClient](#) in a Directory or Location block, then [mod_ssl](#) must buffer any HTTP request body into memory until the new SSL handshake can be performed. This directive can be used to set the amount of memory that will be used for this buffer.

Note that in many configurations, the client sending the request body will be untrusted so a denial of service attack by consumption of memory must be considered when changing this configuration setting.

Example

```
SSLRenegBufferSize 262144
```



Description: Allow access only when an arbitrarily complex boolean expression is true

Syntax: `SSLRequire expression`

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_ssl

This directive specifies a general access requirement which has to be fulfilled in order to allow access. It is a very powerful directive because the requirement specification is an arbitrarily complex boolean expression containing any number of access checks.

The implementation of `SSLRequire` is not thread safe. Using `SSLRequire` inside `.htaccess` files on a threaded [MPM](#) may cause random crashes.

The *expression* must match the following syntax (given as a BNF grammar notation):

```
expr ::= "true" | "false"
      | "!" expr
      | expr "&&" expr
      | expr "||" expr
      | "(" expr ")"
      | comp

comp ::= word "==" word | word "eq" word
      | word "!=" word | word "ne" word
      | word "<" word | word "lt" word
      | word "<=" word | word "le" word
      | word ">" word | word "gt" word
      | word ">=" word | word "ge" word
```

```

        | word "in" "{" wordlist "}"
        | word "in" "OID(" word ")"
        | word "=~" regex
        | word "!~" regex

wordlist ::= word
          | wordlist "," word

word     ::= digit
          | cstring
          | variable
          | function

digit    ::= [0-9]+
cstring  ::= "... "
variable ::= "%{" varname "}"
function ::= funcname "(" funcargs ")"

```

while for varname any variable from [Table 3](#) can be used. Finally for funcname the following functions are available:

- `file(filename)`

This function takes one string argument and expands to the contents of the file. This is especially useful for matching this contents against a regular expression, etc.

Notice that *expression* is first parsed into an internal machine representation and then evaluated in a second step. Actually, in Global and Per-Server Class context *expression* is parsed at startup time and at runtime only the machine representation is executed. For Per-Directory context, specifically in a .htaccess context, this is different: here *expression* has to be parsed and immediately executed for every request.

Example

```

SSLRequire ( %{SSL_CIPHER} !~ m/^(EXP|NULL)-/ \
and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \

```

```

and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"} \
and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20 ) \
or %{REMOTE_ADDR} =~ m/^\192\.76\.162\.[0-9]+$/

```

The `OID()` function expects to find zero or more instances of the given OID in the client certificate, and compares the left-hand side string against the value of matching OID attributes. Every matching OID is checked, until a match is found.

Standard CGI/1.0 and Apache variables:

HTTP_USER_AGENT	PATH_INFO	AUTH_TYPE
HTTP_REFERER	QUERY_STRING	SERVER_SO
HTTP_COOKIE	REMOTE_HOST	API_VERSI
HTTP_FORWARDED	REMOTE_IDENT	TIME_YEAR
HTTP_HOST	IS_SUBREQ	TIME_MON
HTTP_PROXY_CONNECTION	DOCUMENT_ROOT	TIME_DAY
HTTP_ACCEPT	SERVER_ADMIN	TIME_HOUR
HTTP:headername	SERVER_NAME	TIME_MIN
THE_REQUEST	SERVER_PORT	TIME_SEC
REQUEST_METHOD	SERVER_PROTOCOL	TIME_WDAY
REQUEST_SCHEME	REMOTE_ADDR	TIME
REQUEST_URI	REMOTE_USER	ENV:varia
REQUEST_FILENAME		

SSL-related variables:

HTTPS	SSL_CLIENT_M_VERSION	SSL_SERVI
	SSL_CLIENT_M_SERIAL	SSL_SERVI
SSL_PROTOCOL	SSL_CLIENT_V_START	SSL_SERVI
SSL_SESSION_ID	SSL_CLIENT_V_END	SSL_SERVI
SSL_CIPHER	SSL_CLIENT_S_DN	SSL_SERVI
SSL_CIPHER_EXPORT	SSL_CLIENT_S_DN_C	SSL_SERVI
SSL_CIPHER_ALGKEYSIZE	SSL_CLIENT_S_DN_ST	SSL_SERVI
SSL_CIPHER_USEKEYSIZE	SSL_CLIENT_S_DN_L	SSL_SERVI
SSL_VERSION_LIBRARY	SSL_CLIENT_S_DN_O	SSL_SERVI
SSL_VERSION_INTERFACE	SSL_CLIENT_S_DN_OU	SSL_SERVI

SSL_CLIENT_S_DN_CN	SSL_SERVI
SSL_CLIENT_S_DN_T	SSL_SERVI
SSL_CLIENT_S_DN_I	SSL_SERVI
SSL_CLIENT_S_DN_G	SSL_SERVI
SSL_CLIENT_S_DN_S	SSL_SERVI
SSL_CLIENT_S_DN_D	SSL_SERVI
SSL_CLIENT_S_DN_UID	SSL_SERVI
SSL_CLIENT_S_DN_Email	SSL_SERVI
SSL_CLIENT_I_DN	SSL_SERVI
SSL_CLIENT_I_DN_C	SSL_SERVI
SSL_CLIENT_I_DN_ST	SSL_SERVI
SSL_CLIENT_I_DN_L	SSL_SERVI
SSL_CLIENT_I_DN_O	SSL_SERVI
SSL_CLIENT_I_DN_OU	SSL_SERVI
SSL_CLIENT_I_DN_CN	SSL_SERVI
SSL_CLIENT_I_DN_T	SSL_SERVI
SSL_CLIENT_I_DN_I	SSL_SERVI
SSL_CLIENT_I_DN_G	SSL_SERVI
SSL_CLIENT_I_DN_S	SSL_SERVI
SSL_CLIENT_I_DN_D	SSL_SERVI
SSL_CLIENT_I_DN_UID	SSL_SERVI
SSL_CLIENT_I_DN_Email	SSL_SERVI
SSL_CLIENT_A_SIG	SSL_SERVI
SSL_CLIENT_A_KEY	SSL_SERVI
SSL_CLIENT_CERT	SSL_SERVI
SSL_CLIENT_CERT_CHAIN_n	
SSL_CLIENT_VERIFY	SSL_TLS_!



Description:	Deny access when SSL is not used for the HTTP request
Syntax:	SSLRequireSSL
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive forbids access unless HTTP over SSL (i.e. HTTPS) is enabled for the current connection. This is very handy inside the SSL-enabled virtual host or directories for defending against configuration errors that expose stuff that should be protected. When this directive is present all requests are denied which are not using SSL.

Example

```
SSLRequireSSL
```



Description:	Type of the global/inter-process SSL Session Cache
Syntax:	SSLSessionCache <i>type</i>
Default:	SSLSessionCache none
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures the storage type of the global/inter-process SSL Session Cache. This cache is an optional facility which speeds up parallel request processing. For requests to the same server process (via HTTP keep-alive), OpenSSL already caches the SSL session information locally. But because modern clients request inlined images and other data via parallel requests (usually up to four parallel requests are common) those requests are served by *different* pre-forked server processes. Here an inter-process cache helps to avoid unnecessary session handshakes.

The following four storage *types* are currently supported:

- none
This disables the global/inter-process Session Cache. This will incur a noticeable speed penalty and may cause problems if using certain browsers, particularly if client certificates are enabled. This setting is not recommended.
- nonenotnull
This disables any global/inter-process Session Cache. However it does force OpenSSL to send a non-null session ID to accommodate buggy clients that require one.
- dbm:/path/to/datafile
This makes use of a DBM hashfile on the local disk to synchronize the local OpenSSL memory caches of the server processes. This session cache may suffer reliability issues under high load.

- `shm:/path/to/datafile[(size)]`
This makes use of a high-performance cyclic buffer (approx. *size* bytes in size) inside a shared memory segment in RAM (established via `/path/to/datafile`) to synchronize the local OpenSSL memory caches of the server processes. This is the recommended session cache.
- `dc:UNIX:/path/to/socket`
This makes use of the `distcache` distributed session caching libraries. The argument should specify the location of the server or proxy to be used using the `distcache` address syntax; for example, `UNIX:/path/to/socket` specifies a UNIX domain socket (typically a local `dc_client` proxy); `IP:server.example.com:9001` specifies an IP address.

Examples

```
SSLSessionCache dbm:/usr/local/apache/logs/ssl_gcach_data  
SSLSessionCache shm:/usr/local/apache/logs/ssl_gcach_data(512000)
```



Description:	Number of seconds before an SSL session expires in the Session Cache
Syntax:	<code>SSLSessionCacheTimeout</code> <i>seconds</i>
Default:	<code>SSLSessionCacheTimeout</code> 300
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Applies also to RFC 5077 TLS session resumption in Apache 2.2.28 and later

This directive sets the timeout in seconds for the information stored in the global/inter-process SSL Session Cache, the OpenSSL internal memory cache and for sessions resumed by TLS session resumption (RFC 5077). It can be set as low as 15 for testing, but should be set to higher values like 300 in real life.

Example

```
SSLSessionCacheTimeout 600
```



Description:	Persistent encryption/decryption key for TLS session tickets
Syntax:	<code>SSLSessionTicketKeyFile</code> <i>file-path</i>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_ssl</code>
Compatibility:	Available in <code>httpd 2.2.30</code> and later, if using <code>OpenSSL 0.9.8h</code> or later

Optionally configures a secret key for encrypting and decrypting TLS session tickets, as defined in [RFC 5077](#). Primarily suitable for clustered environments where TLS sessions information should be shared between multiple nodes. For single-instance `httpd` setups, it is recommended to *not* configure a ticket key file, but to rely on (random) keys generated by `mod_ssl` at startup, instead.

The ticket key file must contain 48 bytes of random data, preferably created from a high-entropy source. On a Unix-based system, a ticket key file can be created as follows:

```
dd if=/dev/random of=/path/to/file.tkey bs=1 count=48
```

Ticket keys should be rotated (replaced) on a frequent basis, as this is the only way to invalidate an existing session ticket - `OpenSSL` currently doesn't allow to specify a limit for ticket lifetimes. A new ticket key only gets used after restarting the web server. All existing session tickets become invalid after a restart.

The ticket key file contains sensitive keying material and should be protected with file permissions similar to those used for [`SSLCertificateKeyFile`](#).



Description:	Enable or disable use of TLS session tickets
Syntax:	SSLSessionTickets on off
Default:	SSLSessionTickets on
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in httpd 2.2.30 and later, if using OpenSSL 0.9.8f or later.

This directive allows to enable or disable the use of TLS session tickets (RFC 5077).

TLS session tickets are enabled by default. Using them without restarting the web server with an appropriate frequency (e.g. daily) compromises perfect forward secrecy.



Description:	Whether to allow non SNI clients to access a name based virtual host.
Syntax:	SSLStrictSNIVHostCheck on off
Default:	SSLStrictSNIVHostCheck off
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in Apache 2.2.12 and later

This directive sets whether a non SNI client is allowed to access a name based virtual host. If set to on in the non default name based virtual host, non SNI clients are not allowed to access this particular virtual host. If set to on in the default name based virtual host, non SNI clients are not allowed to access any name based virtual host belonging to this IP / port combination.

This option is only available if httpd was compiled against an SNI capable version of OpenSSL.

Example

```
SSLStrictSNIVHostCheck on
```



Description:	Variable name to determine user name
Syntax:	SSLUserName <i>varname</i>
Context:	server config, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl
Compatibility:	Available in Apache 2.0.51 and later

This directive sets the "user" field in the Apache request object. This is used by lower modules to identify the user with a character string. In particular, this may cause the environment variable REMOTE_USER to be set. The *varname* can be any of the [SSL environment variables](#).

Note that this directive has no effect if the FakeBasicAuth option is used (see [SSLOptions](#)).

Example

```
SSLUserName SSL_CLIENT_S_DN_CN
```



Description:	Type of Client Certificate verification
Syntax:	SSLVerifyClient <i>level</i>
Default:	SSLVerifyClient none
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets the Certificate verification level for the Client Authentication. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured client verification level after the HTTP request was read but before the HTTP response is sent.

The following levels are available for *level*:

- **none**: no client Certificate is required at all
- **optional**: the client *may* present a valid Certificate
- **require**: the client *has to* present a valid Certificate
- **optional_no_ca**: the client may present a valid Certificate but it need not to be (successfully) verifiable.

In practice only levels **none** and **require** are really interesting, because level **optional** doesn't work with all browsers and level **optional_no_ca** is actually against the idea of authentication (but can be used to establish SSL test pages, etc.)

Example

```
SSLVerifyClient require
```



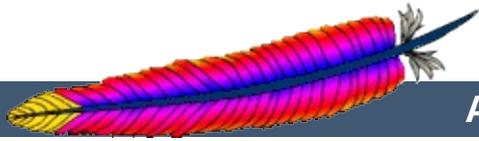
Description:	Maximum depth of CA Certificates in Client Certificate verification
Syntax:	SSLVerifyDepth <i>number</i>
Default:	SSLVerifyDepth 1
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets how deeply mod_ssl should verify before deciding that the clients don't have a valid certificate. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured client verification depth after the HTTP request was read but before the HTTP response is sent.

The depth actually is the maximum number of intermediate certificate issuers, i.e. the number of CA certificates which are max allowed to be followed while verifying the client certificate. A depth of 0 means that self-signed client certificates are accepted only, the default depth of 1 means the client certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA's certificate is under [SSLCACertificatePath](#)), etc.

Example

```
SSLVerifyDepth 10
```



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_status

Description:	Provides information on server activity and performance
Status:	Base
Module Identifier:	status_module
Source File:	mod_status.c

Summary

The Status module allows a server administrator to find out how well their server is performing. A HTML page is presented that gives the current server statistics in an easily readable form. If required this page can be made to automatically refresh (given a compatible browser). Another page gives a simple machine-readable list of the current server state.

The details given are:

- The number of worker serving requests
- The number of idle worker
- The status of each worker, the number of requests that worker has performed and the total number of bytes served by the worker (*)
- A total number of accesses and byte count served (*)
- The time the server was started/restarted and the time it has been running for
- Averages giving the number of requests per second, the number of bytes served per second and the average number of bytes per request (*)
- The current percentage CPU used by each worker and in total by Apache (*)
- The current hosts and requests being processed (*)

The lines marked "*" are only available if ExtendedStatus is On.



Enabling Status Support

To enable status reports only for browsers from the example.com domain add this code to your `httpd.conf` configuration file

```
<Location /server-status>
  SetHandler server-status

  Order Deny,Allow
  Deny from all
  Allow from .example.com
</Location>
```

You can now access server statistics by using a Web browser to access the page `http://your.server.name/server-status`



Automatic Updates

You can get the status page to update itself automatically if you have a browser that supports "refresh". Access the page `http://your.server.name/server-status?refresh=N` to refresh the page every N seconds.



Machine-Readable Status File

A machine-readable version of the status file is available by accessing the page `http://your.server.name/server-status?auto`. This is useful when automatically run, see the Perl program in the `/support` directory of Apache, `log_server_status`.

It should be noted that if `mod_status` is compiled into the server, its handler capability is available in *all* configuration files, including *per-directory* files (e.g., `.htaccess`). This may have security-related ramifications for your site.



Description:	Keep track of extended status information for each request
Syntax:	ExtendedStatus On Off
Default:	ExtendedStatus Off
Context:	server config
Status:	Base
Module:	mod_status
Compatibility:	ExtendedStatus is only available in Apache 1.3.2 and later.

This setting applies to the entire server, and cannot be enabled or disabled on a virtualhost-by-virtualhost basis. The collection of extended status information can slow down the server.



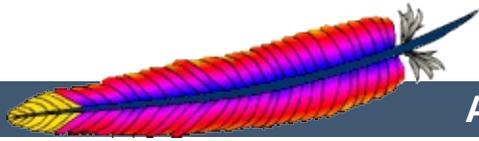
Description:	Determine if <code>mod_status</code> displays the first 63 characters of a request or the last 63, assuming the request itself is greater than 63 chars.
Syntax:	<code>SeeRequestTail On Off</code>
Default:	<code>SeeRequestTail Off</code>
Context:	server config
Status:	Base
Module:	<code>mod_status</code>
Compatibility:	Available in Apache 2.2.7 and later.

`mod_status` with `ExtendedStatus On` displays the actual request being handled. For historical purposes, only 63 characters of the request are actually stored for display purposes. This directive controls whether the 1st 63 characters are stored (the previous behavior and the default) or if the last 63 characters are. This is only applicable, of course, if the length of the request is 64 characters or greater.

If Apache is handling
GET /disk1/storage/apache/htdocs/images/imagestore1/fo
`mod_status` displays as follows:

Off (default)	GET /disk1/storage/apache/htdocs/images/imagestore1/food/a
On	orage/apache/htdocs/images/imagestore1/food/apples.jpg HTT

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_substitute`

Description:	Perform search and replace operations on response bodies
Status:	Extension
Module Identifier:	<code>substitute_module</code>
Source File:	<code>mod_substitute.c</code>
Compatibility:	Available in Apache 2.2.7 and later

Summary

`mod_substitute` provides a mechanism to perform both regular expression and fixed string substitutions on response bodies.



Description:	Pattern to filter the response content
Syntax:	Substitute <i>s/pattern/substitution/[infq]</i>
Context:	directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_substitute

The **Substitute** directive specifies a search and replace pattern to apply to the response body.

The meaning of the pattern can be modified by using any combination of these flags:

i

Perform a case-insensitive match.

n

By default the pattern is treated as a regular expression. Using the **n** flag forces the pattern to be treated as a fixed string.

f

The **f** flag causes `mod_substitute` to flatten the result of a substitution allowing for later substitutions to take place on the boundary of this one. This is the default.

q

The **q** flag causes `mod_substitute` to not flatten the buckets after each substitution. This can result in much faster response and a decrease in memory utilization, but should only be used if there is no possibility that the result of one substitution will ever match a pattern or regex of a subsequent one.

Example

```
<Location />  
  AddOutputFilterByType SUBSTITUTE text/html
```

```
Substitute s/foo/bar/ni
</Location>
```

If either the pattern or the substitution contain a slash character then an alternative delimiter should be used:

Example of using an alternate delimiter

```
<Location />
  AddOutputFilterByType SUBSTITUTE text/html
  Substitute "s|<BR */?>|<br />|i"
</Location>
```

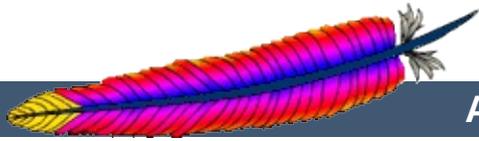


Description:	Change the merge order of inherited patterns
Syntax:	<code>SubstituteInheritBefore on off</code>
Default:	<code>SubstituteInheritBefore off</code>
Context:	directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_substitute
Compatibility:	Available in httpd 2.2.32 and later

Whether to apply the inherited `Substitute` patterns first (on), or after the ones of the current context (off). `SubstituteInheritBefore` is itself inherited, hence contexts that inherit it (those that don't specify their own `SubstituteInheritBefore` value) will apply the closest defined merge order.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_suexec

Description:	Allows CGI scripts to run as a specified user and Group
Status:	Extension
Module Identifier:	suexec_module
Source File:	mod_suexec.c
Compatibility:	Available in Apache 2.0 and later

Summary

This module, in combination with the [suexec](#) support program allows CGI scripts to run as a specified user and Group.

See also

[SuEXEC support](#)



Description:	User and group for CGI programs to run as
Syntax:	<code>SuexecUserGroup <i>User Group</i></code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_suexec</code>
Compatibility:	SuexecUserGroup is only available in 2.0 and later.

The `SuexecUserGroup` directive allows you to specify a user and group for CGI programs to run as. Non-CGI requests are still processed with the user specified in the `User` directive. This directive replaces the Apache 1.3 configuration of using the `User` and `Group` directives inside of `VirtualHosts`.

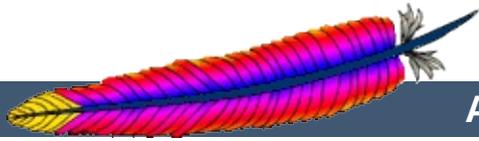
Example

```
SuexecUserGroup nobody nogroup
```

See also

- [Suexec](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_unique_id

Description:	Provides an environment variable with a unique identifier for each request
Status:	Extension
Module Identifier:	unique_id_module
Source File:	mod_unique_id.c

Summary

This module provides a magic token for each request which is guaranteed to be unique across "all" requests under very specific conditions. The unique identifier is even unique across multiple machines in a properly configured cluster of machines. The environment variable `UNIQUE_ID` is set to the identifier for each request. Unique identifiers are useful for various reasons which are beyond the scope of this document.



First a brief recap of how the Apache server works on Unix machines. This feature currently isn't supported on Windows NT. On Unix machines, Apache creates several children, the children process requests one at a time. Each child can serve multiple requests in its lifetime. For the purpose of this discussion, the children don't share any data with each other. We'll refer to the children as *httpd processes*.

Your website has one or more machines under your administrative control, together we'll call them a cluster of machines. Each machine can possibly run multiple instances of Apache. All of these collectively are considered "the universe", and with certain assumptions we'll show that in this universe we can generate unique identifiers for each request, without extensive communication between machines in the cluster.

The machines in your cluster should satisfy these requirements. (Even if you have only one machine you should synchronize its clock with NTP.)

- The machines' times are synchronized via NTP or other network time protocol.
- The machines' hostnames all differ, such that the module can do a hostname lookup on the hostname and receive a different IP address for each machine in the cluster.

As far as operating system assumptions go, we assume that pids (process ids) fit in 32-bits. If the operating system uses more than 32-bits for a pid, the fix is trivial but must be performed in the code.

Given those assumptions, at a single point in time we can identify any httpd process on any machine in the cluster from all other httpd processes. The machine's IP address and the pid of the httpd process are sufficient to do this. A httpd process can handle multiple requests simultaneously if you use a multi-threaded MPM. In order to identify threads, we use a thread index Apache httpd uses internally. So in order to generate unique identifiers for requests we need only distinguish

between different points in time.

To distinguish time we will use a Unix timestamp (seconds since January 1, 1970 UTC), and a 16-bit counter. The timestamp has only one second granularity, so the counter is used to represent up to 65536 values during a single second. The quadruple (*ip_addr*, *pid*, *time_stamp*, *counter*) is sufficient to enumerate 65536 requests per second per httpd process. There are issues however with pid reuse over time, and the counter is used to alleviate this issue.

When an httpd child is created, the counter is initialized with (current microseconds divided by 10) modulo 65536 (this formula was chosen to eliminate some variance problems with the low order bits of the microsecond timers on some systems). When a unique identifier is generated, the time stamp used is the time the request arrived at the web server. The counter is incremented every time an identifier is generated (and allowed to roll over).

The kernel generates a pid for each process as it forks the process, and pids are allowed to roll over (they're 16-bits on many Unixes, but newer systems have expanded to 32-bits). So over time the same pid will be reused. However unless it is reused within the same second, it does not destroy the uniqueness of our quadruple. That is, we assume the system does not spawn 65536 processes in a one second interval (it may even be 32768 processes on some Unixes, but even this isn't likely to happen).

Suppose that time repeats itself for some reason. That is, suppose that the system's clock is screwed up and it revisits a past time (or it is too far forward, is reset correctly, and then revisits the future time). In this case we can easily show that we can get pid and time stamp reuse. The choice of initializer for the counter is intended to help defeat this. Note that we really want a random number to initialize the counter, but there aren't any readily available numbers on most systems (*i.e.*, you can't use `rand()` because you need to seed the generator, and can't seed it with the

time because time, at least at one second resolution, has repeated itself). This is not a perfect defense.

How good a defense is it? Suppose that one of your machines serves at most 500 requests per second (which is a very reasonable upper bound at this writing, because systems generally do more than just shovel out static files). To do that it will require a number of children which depends on how many concurrent clients you have. But we'll be pessimistic and suppose that a single child is able to serve 500 requests per second. There are 1000 possible starting counter values such that two sequences of 500 requests overlap. So there is a 1.5% chance that if time (at one second resolution) repeats itself this child will repeat a counter value, and uniqueness will be broken. This was a very pessimistic example, and with real world values it's even less likely to occur. If your system is such that it's still likely to occur, then perhaps you should make the counter 32 bits (by editing the code).

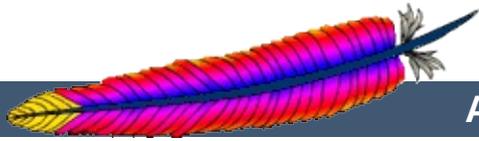
You may be concerned about the clock being "set back" during summer daylight savings. However this isn't an issue because the times used here are UTC, which "always" go forward. Note that x86 based Unices may need proper configuration for this to be true -- they should be configured to assume that the motherboard clock is on UTC and compensate appropriately. But even still, if you're running NTP then your UTC time will be correct very shortly after reboot.

The `UNIQUE_ID` environment variable is constructed by encoding the 144-bit (32-bit IP address, 32 bit pid, 32 bit time stamp, 16 bit counter, 32 bit thread index) quadruple using the alphabet `[A-Za-z0-9@-]` in a manner similar to MIME base64 encoding, producing 24 characters. The MIME base64 alphabet is actually `[A-Za-z0-9+/-]` however `+` and `/` need to be specially encoded in URLs, which makes them less desirable. All values are encoded in network byte ordering so that the encoding is comparable across architectures of different byte ordering. The actual ordering of the encoding is: time stamp, IP address, pid, counter. This ordering has a purpose, but it should be emphasized that applications

should not dissect the encoding. Applications should treat the entire encoded `UNIQUE_ID` as an opaque token, which can be compared against other `UNIQUE_ID`s for equality only.

The ordering was chosen such that it's possible to change the encoding in the future without worrying about collision with an existing database of `UNIQUE_ID`s. The new encodings should also keep the time stamp as the first element, and can otherwise use the same alphabet and bit length. Since the time stamps are essentially an increasing sequence, it's sufficient to have a *flag second* in which all machines in the cluster stop serving any request, and stop using the old encoding format. Afterwards they can resume requests and begin issuing the new encodings.

This we believe is a relatively portable solution to this problem. The identifiers generated have essentially an infinite life-time because future identifiers can be made longer as required. Essentially no communication is required between machines in the cluster (only NTP synchronization is required, which is low overhead), and no communication between httpd processes is required (the communication is implicit in the pid value assigned by the kernel). In very specific situations the identifier can be shortened, but more information needs to be assumed (for example the 32-bit IP address is overkill for any site, but there is no portable shorter replacement for it).



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_userdir

Description:	User-specific directories
Status:	Base
Module Identifier:	userdir_module
Source File:	mod_userdir.c

Summary

This module allows user-specific directories to be accessed using the `http://example.com/~user/` syntax.

See also

[Mapping URLs to the Filesystem](#)
[public_html tutorial](#)



Description:	Location of the user-specific directories
Syntax:	UserDir <i>directory-filename</i> [<i>directory-filename</i>] ...
Context:	server config, virtual host
Status:	Base
Module:	mod_userdir

The **UserDir** directive sets the real directory in a user's home directory to use when a request for a document for a user is received. *Directory-filename* is one of the following:

- The name of a directory or a pattern such as those shown below.
- The keyword `disabled`. This turns off *all* username-to-directory translations except those explicitly named with the `enabled` keyword (see below).
- The keyword `disabled` followed by a space-delimited list of usernames. Usernames that appear in such a list will *never* have directory translation performed, even if they appear in an `enabled` clause.
- The keyword `enabled` followed by a space-delimited list of usernames. These usernames will have directory translation performed even if a global `disable` is in effect, but not if they also appear in a `disabled` clause.

If neither the `enabled` nor the `disabled` keywords appear in the `Userdir` directive, the argument is treated as a filename pattern, and is used to turn the name into a directory specification. A request for `http://www.example.com/~bob/one/two.html` will be translated to:

UserDir directive used	Translated path
UserDir public_html	~bob/public_html/one/two.html

```
UserDir /usr/web      /usr/web/bob/one/two.html
UserDir /home/*/www   /home/bob/www/one/two.html
```

The following directives will send redirects to the client:

UserDir directive used	Translated path
UserDir http://www.example.com/users	http://www.example.com/users/bob/one/two.html
UserDir http://www.example.com/*/usr	http://www.example.com/bob/usr/one/two.html
UserDir http://www.example.com/~*/	http://www.example.com/~bob/one/two.html

Be careful when using this directive; for instance, "UserDir ./" would map "/~root" to "/" - which is probably undesirable. It is strongly recommended that your configuration include a "UserDir disabled root" declaration. See also the [Directory](#) directive and the [Security Tips](#) page for more information.

Additional examples:

To allow a few users to have UserDir directories, but not anyone else, use the following:

```
UserDir disabled
UserDir enabled user1 user2 user3
```

To allow most users to have UserDir directories, but deny this to a few, use the following:

```
UserDir disabled user4 user5 user6
```

It is also possible to specify alternative user directories. If you use a command like:

```
Userdir public_html /usr/web http://www.example.com/
```

With a request for `http://www.example.com/~bob/one/two.html`, will try to find the page at `~bob/public_html/one/two.html` first, then `/usr/web/bob/one/two.html`, and finally it will send a redirect to `http://www.example.com/bob/one/two.html`.

If you add a redirect, it must be the last alternative in the list. Apache cannot determine if the redirect succeeded or not, so if you have the redirect earlier in the list, that will always be the alternative that is used.

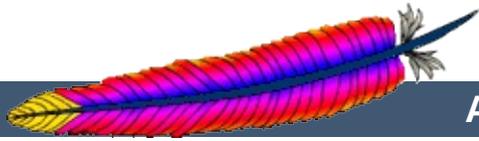
User directory substitution is not active by default in versions 2.1.4 and later. In earlier versions, `UserDir public_html` was assumed if no `UserDir` directive was present.

See also

- [public_html tutorial](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_usertrack

Description:	<i>Clickstream</i> logging of user activity on a site
Status:	Extension
Module Identifier:	usertrack_module
Source File:	mod_usertrack.c

Summary

Previous releases of Apache have included a module which generates a 'clickstream' log of user activity on a site using cookies. This was called the "cookies" module, mod_cookies. In Apache 1.2 and later this module has been renamed the "user tracking" module, mod_usertrack. This module has been simplified and new directives added.



Previously, the cookies module (now the user tracking module) did its own logging, using the `CookieLog` directive. In this release, this module does no logging at all. Instead, a configurable log format file should be used to log user click-streams. This is possible because the logging module now allows multiple log files. The cookie itself is logged by using the text `%{cookie}`n in the log file format. For example:

```
CustomLog logs/clickstream "%{cookie}n %r %t"
```

For backward compatibility the configurable log module implements the old `CookieLog` directive, but this should be upgraded to the above `CustomLog` directive.



(the following is from message

<022701bda43d\$9d32bbb0\$1201a8c0@christian.office.sane.com> in the new-httpd archives)

From: "Christian Allen" <christian@sane.com>
Subject: Re: Apache Y2K bug in mod_usertrack.c
Date: Tue, 30 Jun 1998 11:41:56 -0400

Did some work with cookies and dug up some info that m.

True, Netscape claims that the correct format NOW is four digit dates do in fact work... for Netscape 4.x (is. However, 3.x and below do NOT accept them. It seems originally had a 2-digit standard, and then with all of probably a few complaints, changed to a four digit date. Fortunately, 4.x also understands the 2-digit format, ensure that your expiration date is legible to the client use 2-digit dates.

However, this does not limit expiration dates to the year an expiration year of "13", for example, it is interpreted as 1913! In fact, you can use an expiration year of up to 2037 understood as "2037" by both MSIE and Netscape versions (about versions previous to those). Not sure why Netscape chose a particular year as its cut-off point, but my guess is related to UNIX's 2038 problem. Netscape/MSIE 4.x seem to be able to handle 2-digit years beyond that, at least until "50" for sure (I can understand up until about "70", but not for sure).

Summary: Mozilla 3.x and up understands two digit dates (2037). Mozilla 4.x understands up until at least "50" in that form, but also understands 4-digit years, which can probably go to 9999. Your best bet for sending a long-life cookie is to set the time late in the year "37".



Description:	The domain to which the tracking cookie applies
Syntax:	CookieDomain <i>domain</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_usertrack

This directive controls the setting of the domain to which the tracking cookie applies. If not present, no domain is included in the cookie header field.

The domain string **must** begin with a dot, and **must** include at least one embedded dot. That is, `.example.com` is legal, but `foo.example.com` and `.com` are not.

Most browsers in use today will not allow cookies to be set for a two-part top level domain, such as `.co.uk`, although such a domain ostensibly fulfills the requirements above.

These domains are equivalent to top level domains such as `.com`, and allowing such cookies may be a security risk. Thus, if you are under a two-part top level domain, you should still use your actual domain, as you would with any other top level domain (for example, use `.foo.co.uk`).



Description:	Expiry time for the tracking cookie
Syntax:	CookieExpires <i>expiry-period</i>
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_usertrack

When used, this directive sets an expiry time on the cookie generated by the usertrack module. The *expiry-period* can be given either as a number of seconds, or in the format such as "2 weeks 3 days 7 hours". Valid denominations are: years, months, weeks, days, hours, minutes and seconds. If the expiry time is in any format other than one number indicating the number of seconds, it must be enclosed by double quotes.

If this directive is not used, cookies last only for the current browser session.



Description:	Name of the tracking cookie
Syntax:	CookieName <i>token</i>
Default:	CookieName Apache
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_usertrack

This directive allows you to change the name of the cookie this module uses for its tracking purposes. By default the cookie is named "Apache".

You must specify a valid cookie name; results are unpredictable if you use a name containing unusual characters. Valid characters include A-Z, a-z, 0-9, "_", and "-".



Description:	Format of the cookie header field
Syntax:	CookieStyle <i>Netscape Cookie Cookie2 RFC2109 RFC2965</i>
Default:	CookieStyle Netscape
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_usertrack

This directive controls the format of the cookie header field. The three formats allowed are:

- **Netscape**, which is the original but now deprecated syntax. This is the default, and the syntax Apache has historically used.
- **Cookie** or **RFC2109**, which is the syntax that superseded the Netscape syntax.
- **Cookie2** or **RFC2965**, which is the most current cookie syntax.

Not all clients can understand all of these formats, but you should use the newest one that is generally acceptable to your users' browsers. At the time of writing, most browsers only fully support CookieStyle Netscape.

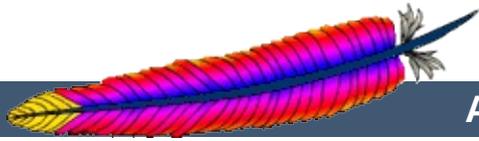


Description:	Enables tracking cookie
Syntax:	CookieTracking on off
Default:	CookieTracking off
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_usertrack

When [mod_usertrack](#) is loaded, and `CookieTracking on` is set, Apache will send a user-tracking cookie for all new requests. This directive can be used to turn this behavior on or off on a per-server or per-directory basis. By default, enabling [mod_usertrack](#) will **not** activate cookies.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module mod_version

Description:	Version dependent configuration
Status:	Extension
Module Identifier:	version_module
Source File:	mod_version.c
Compatibility:	Available in version 2.0.56 and later

Summary

This module is designed for the use in test suites and large networks which have to deal with different httpd versions and different configurations. It provides a new container -- `<IfVersion>`, which allows a flexible version checking including numeric comparisons and regular expressions.

Examples

```
<IfVersion 2.1.0>
  # current httpd version is exactly 2.1.0
</IfVersion>

<IfVersion >= 2.2>
  # use really new features :-)
</IfVersion>
```

See below for further possibilities.



Description:	contains version dependent configuration
Syntax:	<code><IfVersion [[!]operator] version> ... </IfVersion></code>
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Extension
Module:	mod_version

The `<IfVersion>` section encloses configuration directives which are executed only if the `httpd` version matches the desired criteria. For normal (numeric) comparisons the *version* argument has the format *major* [*.minor* [*.patch*]], e.g. 2.1.0 or 2.2. *minor* and *patch* are optional. If these numbers are omitted, they are assumed to be zero. The following numerical *operators* are possible:

<i>operator</i>	<i>description</i>
= or ==	httpd version is equal
>	httpd version is greater than
>=	httpd version is greater or equal
<	httpd version is less than
<=	httpd version is less or equal

Example

```
<IfVersion >= 2.1>
  # this happens only in versions greater or
  # equal 2.1.0.
</IfVersion>
```

Besides the numerical comparison it is possible to match a [regular expression](#) against the `httpd` version. There are two ways to write it:

<i>operator</i>	<i>description</i>
-----------------	--------------------

= or ==	<i>version</i> has the form <i>/regex/</i>
~	<i>version</i> has the form <i>regex</i>

Example

```
<IfVersion = /^2.1.[01234]$/>  
  # e.g. workaround for buggy versions  
</IfVersion>
```

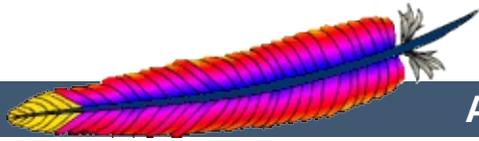
In order to reverse the meaning, all operators can be preceded by an exclamation mark (!):

```
<IfVersion !~ ^2.1.[01234]$/>  
  # not for those versions  
</IfVersion>
```

If the *operator* is omitted, it is assumed to be =.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Modules](#)

Apache Module `mod_vhost_alias`

Description:	Provides for dynamically configured mass virtual hosting
Status:	Extension
Module Identifier:	<code>vhost_alias_module</code>
Source File:	<code>mod_vhost_alias.c</code>

Summary

This module creates dynamically configured virtual hosts, by allowing the IP address and/or the `Host` header of the HTTP request to be used as part of the pathname to determine what files to serve. This allows for easy use of a huge number of virtual hosts with similar configurations.

Note

If `mod_alias` or `mod_userdir` are used for translating URIs to filenames, they will override the directives of `mod_vhost_alias` described below. For example, the following configuration will map `/cgi-bin/script.pl` to `/usr/local/apache2/cgi-bin/bin/script.pl` in all cases:

```
ScriptAlias /cgi-bin/ /usr/local/apache2/cgi-bin/  
VirtualScriptAlias /never/found/%0/cgi-bin/
```

See also

[UseCanonicalName](#)

[Dynamically configured mass virtual hosting](#)



Directory Name Interpolation

All the directives in this module interpolate a string into a pathname. The interpolated string (henceforth called the "name") may be either the server name (see the [UseCanonicalName](#) directive for details on how this is determined) or the IP address of the virtual host on the server in dotted-quad format. The interpolation is controlled by specifiers inspired by `printf` which have a number of formats:

%%	insert a %
%p	insert the port number of the virtual host
%N.M	insert (part of) the name

N and M are used to specify substrings of the name. N selects from the dot-separated components of the name, and M selects characters within whatever N has selected. M is optional and defaults to zero if it isn't present; the dot must be present if and only if M is present. The interpretation is as follows:

0	the whole name
1	the first part
2	the second part
-1	the last part
-2	the penultimate part
2+	the second and all subsequent parts
-2+	the penultimate and all preceding parts
1+ and -1+	the same as 0

If N or M is greater than the number of parts available a single underscore is interpolated.



For simple name-based virtual hosts you might use the following directives in your server configuration file:

```
UseCanonicalName Off
VirtualDocumentRoot /usr/local/apache/vhosts/%0
```

A request for `http://www.example.com/directory/file.html` will be satisfied by the file `/usr/local/apache/vhosts/www.example.com/directory/file.html`

For a very large number of virtual hosts it is a good idea to arrange the files to reduce the size of the vhosts directory. To do this you might use the following in your configuration file:

```
UseCanonicalName Off
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.1/%2.2/%2.3/%2
```

A request for `http://www.domain.example.com/directory/file.html` will be satisfied by the file `/usr/local/apache/vhosts/example.com/d/o/m/domain/directory/file.html`

A more even spread of files can be achieved by hashing from the end of the name, for example:

```
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.-1/%2.-2/%2.-3/%2
```

The example request would come from `/usr/local/apache/vhosts/example.com/n/i/a/domain/directory/file.html`

Alternatively you might use:

```
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.1/%2.2/%2.3/%2.4+
```

The example request would come from
`/usr/local/apache/vhosts/example.com/d/o/m/ain/directo`

For IP-based virtual hosting you might use the following in your configuration file:

```
UseCanonicalName DNS
VirtualDocumentRootIP /usr/local/apache/vhosts/%1/%2/%3/%4/docs
VirtualScriptAliasIP /usr/local/apache/vhosts/%1/%2/%3/%4/cgi-bin
```

A request for
`http://www.domain.example.com/directory/file.html` would be satisfied by the file
`/usr/local/apache/vhosts/10/20/30/40/docs/directory/fi`
if the IP address of `www.domain.example.com` were `10.20.30.40`. A request for `http://www.domain.example.com/cgi-bin/script.pl` would be satisfied by executing the program
`/usr/local/apache/vhosts/10/20/30/40/cgi-bin/script.pl`.

If you want to include the `.` character in a `VirtualDocumentRoot` directive, but it clashes with a `%` directive, you can work around the problem in the following way:

```
VirtualDocumentRoot /usr/local/apache/vhosts/%2.0.%3.0
```

A request for
`http://www.domain.example.com/directory/file.html` will be satisfied by the file
`/usr/local/apache/vhosts/domain.example/directory/file`

The [LogFormat](#) directives `%V` and `%A` are useful in conjunction with this module.



Description:	Dynamically configure the location of the document root for a given virtual host
Syntax:	<code>VirtualDocumentRoot <i>interpolated-directory</i> none</code>
Default:	<code>VirtualDocumentRoot none</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_vhost_alias</code>

The `VirtualDocumentRoot` directive allows you to determine where Apache will find your documents based on the value of the server name. The result of expanding *interpolated-directory* is used as the root of the document tree in a similar manner to the `DocumentRoot` directive's argument. If *interpolated-directory* is none then `VirtualDocumentRoot` is turned off. This directive cannot be used in the same context as `VirtualDocumentRootIP`.



Description:	Dynamically configure the location of the document root for a given virtual host
Syntax:	<code>VirtualDocumentRootIP</code> <i>interpolated-directory</i> none
Default:	<code>VirtualDocumentRootIP</code> none
Context:	server config, virtual host
Status:	Extension
Module:	mod_vhost_alias

The `VirtualDocumentRootIP` directive is like the `VirtualDocumentRoot` directive, except that it uses the IP address of the server end of the connection for directory interpolation instead of the server name.



VirtualScriptAlias Directive

Description:	Dynamically configure the location of the CGI directory for a given virtual host
Syntax:	<code>VirtualScriptAlias <i>interpolated-directory</i> none</code>
Default:	<code>VirtualScriptAlias none</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_vhost_alias</code>

The `VirtualScriptAlias` directive allows you to determine where Apache will find CGI scripts in a similar manner to `VirtualDocumentRoot` does for other documents. It matches requests for URIs starting `/cgi-bin/`, much like `ScriptAlias /cgi-bin/` would.

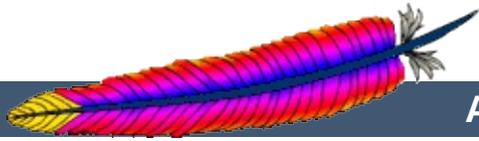


Description:	Dynamically configure the location of the cgi directory for a given virtual host
Syntax:	<code>VirtualScriptAliasIP <i>interpolated-directory</i> none</code>
Default:	<code>VirtualScriptAliasIP none</code>
Context:	server config, virtual host
Status:	Extension
Module:	<code>mod_vhost_alias</code>

The `VirtualScriptAliasIP` directive is like the `VirtualScriptAlias` directive, except that it uses the IP address of the server end of the connection for directory interpolation instead of the server name.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

Apache 1.3 API notes

Warning

This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

These are some notes on the Apache API and the data structures you have to deal with, *etc.* They are not yet nearly complete, but hopefully, they will help you get your bearings. Keep in mind that the API is still subject to change as we gain experience with it. (See the TODO file for what *might* be coming). However, it will be easy to adapt modules to any changes that are made. (We have more modules to adapt than you do).

A few notes on general pedagogical style here. In the interest of conciseness, all structure declarations here are incomplete -- the real ones have more slots that I'm not telling you about. For the most part, these are reserved to one component of the server core or another, and should be altered by modules with caution. However, in some cases, they really are things I just haven't gotten around to yet. Welcome to the bleeding edge.

Finally, here's an outline, to give you some bare idea of what's coming up, and in what order:

- [Basic concepts.](#)
 - [Handlers, Modules, and Requests](#)
 - [A brief tour of a module](#)
- [How handlers work](#)
 - [A brief tour of the request_rec](#)
 - [Where request_rec structures come from](#)

- [Handling requests, declining, and returning error codes](#)
- [Special considerations for response handlers](#)
- [Special considerations for authentication handlers](#)
- [Special considerations for logging handlers](#)

- [Resource allocation and resource pools](#)
- [Configuration, commands and the like](#)
 - [Per-directory configuration structures](#)
 - [Command handling](#)
 - [Side notes --- per-server configuration, virtual servers, etc.](#)



Basic Concepts

We begin with an overview of the basic concepts behind the API, and how they are manifested in the code.

Handlers, Modules, and Requests

Apache breaks down request handling into a series of steps, more or less the same way the Netscape server API does (although this API has a few more stages than NetSite does, as hooks for stuff I thought might be useful in the future). 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' -- there aren't any of these yet, but the phase is intended as a hook for possible extensions like [SetEnv](#), which don't really fit well elsewhere.
- 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 that it has done so by returning the magic constant OK.
- *Decline* to handle the request, by returning the magic integer constant DECLINED. In this case, the server behaves in all respects as if the handler simply hadn't been there.
- Signal an error, by returning one of the HTTP error codes. This terminates normal handling of the request, although an `ErrorDocument` may be invoked to try to mop up, and it will be

logged in any case.

Most phases are terminated by the first module that handles them; however, for logging, 'fixups', and non-access authentication checking, all handlers always run (barring an error). Also, 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). However, 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 `request_rec` structure. vide infra), which returns an integer, as above.

A brief tour of a module

At this point, we need to explain the structure of a module. Our candidate will be one of the messier ones, the CGI module -- this handles both CGI scripts and the `ScriptAlias` config file command. It's actually a great deal more complicated than most modules, but if we're going to have only one example, it might as well be the one with its fingers in every place.

Let's begin with handlers. In order to handle the CGI scripts, the module declares a response handler for them. Because of `ScriptAlias`, it also has handlers for the name translation phase (to recognize `ScriptAliased` URIs), the type-checking phase (any `ScriptAliased` request is typed as a CGI script).

The module needs to maintain some per (virtual) server information, namely, the `ScriptAliases` in effect; the module structure therefore contains pointers to a functions which builds these structures, and to another which combines two of them (in case the main server and a

virtual server both have [ScriptAliases](#) declared).

Finally, this module contains code to handle the [ScriptAlias](#) command itself. This particular module only declares one command, but there could be more, so modules have *command tables* which declare their commands, and describe where they are permitted, and how they are to be invoked.

A final note on the declared types of the arguments of some of these commands: a *pool* is a pointer to a *resource pool* structure; these are used by the server to keep track of the memory which has been allocated, files opened, *etc.*, either to service a particular request, or to handle the process of configuring itself. That way, when the request is over (or, for the configuration pool, when the server is restarting), the memory can be freed, and the files closed, *en masse*, without anyone having to write explicit code to track them all down and dispose of them. Also, a *cmd_parms* structure contains various information about the config file being read, and other status information, which is sometimes of use to the function which processes a config-file command (such as [ScriptAlias](#)). With no further ado, the module itself:

```
/* Declarations of handlers. */

int translate_scriptalias (request_rec *);
int type_scriptalias (request_rec *);
int cgi_handler (request_rec *);

/* Subsidiary dispatch table for response-phase
 * handlers, by MIME type */

handler_rec cgi_handlers[] = {
    { "application/x-httpd-cgi", cgi_handler },
    { NULL }
};

/* Declarations of routines to manipulate the
 * module's configuration info. Note that these are
 * returned, and passed in, as void *'s; the server
 * core keeps track of them, but it doesn't, and can't,
 * know their internal structure.
```

```

*/

void *make_cgi_server_config (pool *);
void *merge_cgi_server_config (pool *, void *, void *);

/* Declarations of routines to handle config-file commands */

extern char *script_alias(cmd_parms *, void *per_dir_config, char
*fake, char *real);

command_rec cgi_cmds[] = {
    { "ScriptAlias", script_alias, NULL, RSRC_CONF, TAKE2,
      "a fakename and a realname"},
    { NULL }
};

module cgi_module = {
    STANDARD_MODULE_STUFF,
    NULL, /* initializer */
    NULL, /* dir config creator */
    NULL, /* dir merger */
    make_cgi_server_config, /* server config */
    merge_cgi_server_config, /* merge server config */
    cgi_cmds, /* command table */
    cgi_handlers, /* handlers */
    translate_scriptalias, /* filename translation */
    NULL, /* check_user_id */
    NULL, /* check_auth */
    NULL, /* check_access */
    type_scriptalias, /* type_checker */
    NULL, /* fixups */
    NULL, /* logger */
    NULL /* header parser */
};

```



The sole argument to handlers is a `request_rec` structure. This structure describes a particular request which has been made to the server, on behalf of a client. In most cases, each connection to the client generates only one `request_rec` structure.

A brief tour of the `request_rec`

The `request_rec` contains pointers to a resource pool which will be cleared when the server is finished handling the request; to structures containing per-server and per-connection information, and most importantly, information on the request itself.

The most important such information is a small set of character strings describing attributes of the object being requested, including its URI, filename, content-type and content-encoding (these being filled in by the translation and type-check handlers which handle the request, respectively).

Other commonly used data items are tables giving the MIME headers on the client's original request, MIME headers to be sent back with the response (which modules can add to at will), and environment variables for any subprocesses which are spawned off in the course of servicing the request. These tables are manipulated using the `ap_table_get` and `ap_table_set` routines.

Note that the Content - type header value *cannot* be set by module content-handlers using the `ap_table_*()` routines. Rather, it is set by pointing the `content_type` field in the `request_rec` structure to an appropriate string. *e.g.*,

```
r->content_type = "text/html";
```

Finally, there are pointers to two data structures which, in turn, point to per-module configuration structures. Specifically, these hold pointers to the data structures which the module has built to describe the way it has been configured to operate in a given directory (via `.htaccess` files or [<Directory>](#) sections), for private data it has built in the course of servicing the request (so modules' handlers for one phase can pass `notes' to their handlers for other phases). There is another such configuration vector in the `server_rec` data structure pointed to by the `request_rec`, which contains per (virtual) server configuration data.

Here is an abridged declaration, giving the fields most commonly used:

```
struct request_rec {

    pool *pool;
    conn_rec *connection;
    server_rec *server;

    /* What object is being requested */

    char *uri;
    char *filename;
    char *path_info;
    char *args;          /* QUERY_ARGS, if any */
    struct stat finfo;   /* Set by server core;
                        * st_mode set to zero if no such file */
    char *content_type;
    char *content_encoding;

    /* MIME header environments, in and out. Also,
     * an array containing environment variables to
     * be passed to subprocesses, so people can write
     * modules to add to that environment.
     *
     * The difference between headers_out and
     * err_headers_out is that the latter are printed
     * even on error, and persist across internal
     * redirects (so the headers printed for
     * ErrorDocument handlers will have them).
     */

    table *headers_in;
```

```

table *headers_out;
table *err_headers_out;
table *subprocess_env;

/* Info about the request itself... */

int header_only;      /* HEAD request, as opposed to GET */
char *protocol;      /* Protocol, as given to us, or HTTP/0.9 */
char *method;        /* GET, HEAD, POST, etc. */
int method_number;   /* M_GET, M_POST, etc. */
/* Info for logging */

char *the_request;
int bytes_sent;

/* A flag which modules can set, to indicate that
 * the data being returned is volatile, and clients
 * should be told not to cache it.
 */

int no_cache;

/* Various other config info which may change
 * with .htaccess files
 * These are config vectors, with one void*
 * pointer for each module (the thing pointed
 * to being the module's business).
 */

void *per_dir_config; /* Options set in config files, etc. */
void *request_config; /* Notes on *this* request */
};

```

Where request_rec structures come from

Most request_rec structures are built by reading an HTTP request from a client, and filling in the fields. However, there are a few exceptions:

- If the request is to an imagemap, a type map (*i.e.*, a *.var file), or a CGI script which returned a local `Location:', then the resource which the user requested is going to be ultimately located by some URI other than what the client originally supplied. In this case, the server

does an *internal redirect*, constructing a new `request_rec` for the new URI, and processing it almost exactly as if the client had requested the new URI directly.

- If some handler signaled an error, and an `ErrorDocument` is in scope, the same internal redirect machinery comes into play.
- Finally, a handler occasionally needs to investigate 'what would happen if' some other request were run. For instance, the directory indexing module needs to know what MIME type would be assigned to a request for each directory entry, in order to figure out what icon to use.

Such handlers can construct a *sub-request*, using the functions `ap_sub_req_lookup_file`, `ap_sub_req_lookup_uri`, and `ap_sub_req_method_uri`; these construct a new `request_rec` structure and processes it as you would expect, up to but not including the point of actually sending a response. (These functions skip over the access checks if the sub-request is for a file in the same directory as the original request).

(Server-side includes work by building sub-requests and then actually invoking the response handler for them, via the function `ap_run_sub_req`).

Handling requests, declining, and returning error codes

As discussed above, each handler, when invoked to handle a particular `request_rec`, has to return an `int` to indicate what happened. That can either be

- OK -- the request was handled successfully. This may or may not terminate the phase.
- DECLINED -- no erroneous condition exists, but the module declines to handle the phase; the server tries to find another.
- an HTTP error code, which aborts handling of the request.

Note that if the error code returned is REDIRECT, then the module should put a Location in the request's headers_out, to indicate where the client should be redirected to.

Special considerations for response handlers

Handlers for most phases do their work by simply setting a few fields in the request_rec structure (or, in the case of access checkers, simply by returning the correct error code). However, response handlers have to actually send a request back to the client.

They should begin by sending an HTTP response header, using the function ap_send_http_header. (You don't have to do anything special to skip sending the header for HTTP/0.9 requests; the function figures out on its own that it shouldn't do anything). If the request is marked header_only, that's all they should do; they should return after that, without attempting any further output.

Otherwise, they should produce a request body which responds to the client as appropriate. The primitives for this are ap_rputc and ap_rprintf, for internally generated output, and ap_send_fd, to copy the contents of some FILE * straight to the client.

At this point, you should more or less understand the following piece of code, which is the handler which handles GET requests which have no more specific handler; it also shows how conditional GETs can be handled, if it's desirable to do so in a particular response handler -- ap_set_last_modified checks against the If-modified-since value supplied by the client, if any, and returns an appropriate code (which will, if nonzero, be USE_LOCAL_COPY). No similar considerations apply for ap_set_content_length, but it returns an error code for symmetry.

```
int default_handler (request_rec *r)
{
```

```

int errstatus;
FILE *f;

if (r->method_number != M_GET) return DECLINED;
if (r->finfo.st_mode == 0) return NOT_FOUND;

if ((errstatus = ap_set_content_length (r, r->finfo.st_size))
    || (errstatus = ap_set_last_modified (r, r->finfo.st_mtime)))
return errstatus;

f = fopen (r->filename, "r");

if (f == NULL) {
    log_reason("file permissions deny server access", r->filename,
        r);
    return FORBIDDEN;
}

register_timeout ("send", r);
ap_send_http_header (r);

if (!r->header_only) send_fd (f, r);
ap_pfclose (r->pool, f);
return OK;
}

```

Finally, if all of this is too much of a challenge, there are a few ways out of it. First off, as shown above, a response handler which has not yet produced any output can simply return an error code, in which case the server will automatically produce an error response. Secondly, it can punt to some other handler by invoking `ap_internal_redirect`, which is how the internal redirection machinery discussed above is invoked. A response handler which has internally redirected should always return OK.

(Invoking `ap_internal_redirect` from handlers which are *not* response handlers will lead to serious confusion).

Special considerations for authentication handlers

Stuff that should be discussed here in detail:

- Authentication-phase handlers not invoked unless auth is configured for the directory.
- Common auth configuration stored in the core per-dir configuration; it has accessors `ap_auth_type`, `ap_auth_name`, and `ap_requires`.
- Common routines, to handle the protocol end of things, at least for HTTP basic authentication (`ap_get_basic_auth_pw`, which sets the `connection->user` structure field automatically, and `ap_note_basic_auth_failure`, which arranges for the proper `WWW-Authenticate:` header to be sent back).

Special considerations for logging handlers

When a request has internally redirected, there is the question of what to log. Apache handles this by bundling the entire chain of redirects into a list of `request_rec` structures which are threaded through the `r->prev` and `r->next` pointers. The `request_rec` which is passed to the logging handlers in such cases is the one which was originally built for the initial request from the client; note that the `bytes_sent` field will only be correct in the last request in the chain (the one for which a response was actually sent).



One of the problems of writing and designing a server-pool server is that of preventing leakage, that is, allocating resources (memory, open files, *etc.*), without subsequently releasing them. The resource pool machinery is designed to make it easy to prevent this from happening, by allowing resource to be allocated in such a way that they are *automatically* released when the server is done with them.

The way this works is as follows: the memory which is allocated, file opened, *etc.*, to deal with a particular request are tied to a *resource pool* which is allocated for the request. The pool is a data structure which itself tracks the resources in question.

When the request has been processed, the pool is *cleared*. At that point, all the memory associated with it is released for reuse, all files associated with it are closed, and any other clean-up functions which are associated with the pool are run. When this is over, we can be confident that all the resource tied to the pool have been released, and that none of them have leaked.

Server restarts, and allocation of memory and resources for per-server configuration, are handled in a similar way. There is a *configuration pool*, which keeps track of resources which were allocated while reading the server configuration files, and handling the commands therein (for instance, the memory that was allocated for per-server module configuration, log files and other files that were opened, and so forth). When the server restarts, and has to reread the configuration files, the configuration pool is cleared, and so the memory and file descriptors which were taken up by reading them the last time are made available for reuse.

It should be noted that use of the pool machinery isn't generally obligatory, except for situations like logging handlers, where you really need to register cleanups to make sure that the log file gets closed when the server restarts (this is most easily done by using the function

[ap_pfopen](#), which also arranges for the underlying file descriptor to be closed before any child processes, such as for CGI scripts, are execed), or in case you are using the timeout machinery (which isn't yet even documented here). However, there are two benefits to using it: resources allocated to a pool never leak (even if you allocate a scratch string, and just forget about it); also, for memory allocation, `ap_palloc` is generally faster than `malloc`.

We begin here by describing how memory is allocated to pools, and then discuss how other resources are tracked by the resource pool machinery.

Allocation of memory in pools

Memory is allocated to pools by calling the function `ap_palloc`, which takes two arguments, one being a pointer to a resource pool structure, and the other being the amount of memory to allocate (in chars). Within handlers for handling requests, the most common way of getting a resource pool structure is by looking at the `pool` slot of the relevant `request_rec`; hence the repeated appearance of the following idiom in module code:

```
int my_handler(request_rec *r)
{
    struct my_structure *foo;
    ...

    foo = (foo *)ap_palloc (r->pool, sizeof(my_structure));
}
```

Note that *there is no `ap_pfree`* -- `ap_palloc`d memory is freed only when the associated resource pool is cleared. This means that `ap_palloc` does not have to do as much accounting as `malloc()`; all it does in the typical case is to round up the size, bump a pointer, and do a range check.

(It also raises the possibility that heavy use of `ap_palloc` could cause a

server process to grow excessively large. There are two ways to deal with this, which are dealt with below; briefly, you can use `malloc`, and try to be sure that all of the memory gets explicitly freed, or you can allocate a sub-pool of the main pool, allocate your memory in the sub-pool, and clear it out periodically. The latter technique is discussed in the section on sub-pools below, and is used in the directory-indexing code, in order to avoid excessive storage allocation when listing directories with thousands of files).

Allocating initialized memory

There are functions which allocate initialized memory, and are frequently useful. The function `ap_pcalloc` has the same interface as `ap_palloc`, but clears out the memory it allocates before it returns it. The function `ap_pstrdup` takes a resource pool and a `char *` as arguments, and allocates memory for a copy of the string the pointer points to, returning a pointer to the copy. Finally `ap_pstrcat` is a varargs-style function, which takes a pointer to a resource pool, and at least two `char *` arguments, the last of which must be `NULL`. It allocates enough memory to fit copies of each of the strings, as a unit; for instance:

```
ap_pstrcat (r->pool, "foo", "/", "bar", NULL);
```

returns a pointer to 8 bytes worth of memory, initialized to "foo/bar".

Commonly-used pools in the Apache Web server

A pool is really defined by its lifetime more than anything else. There are some static pools in `http_main` which are passed to various non-`http_main` functions as arguments at opportune times. Here they are:

permanent_pool

never passed to anything else, this is the ancestor of all pools

pconf

- subpool of permanent_pool
- created at the beginning of a config "cycle"; exists until the server is terminated or restarts; passed to all config-time routines, either via cmd->pool, or as the "pool *p" argument on those which don't take pools
- passed to the module init() functions

p-temp

- sorry I lie, this pool isn't called this currently in 1.3, I renamed it this in my pthreads development. I'm referring to the use of ptrans in the parent... contrast this with the later definition of ptrans in the child.
- subpool of permanent_pool
- created at the beginning of a config "cycle"; exists until the end of config parsing; passed to config-time routines *via* cmd->temp_pool. Somewhat of a "bastard child" because it isn't available everywhere. Used for temporary scratch space which may be needed by some config routines but which is deleted at the end of config.

p-child

- subpool of permanent_pool
- created when a child is spawned (or a thread is created); lives until that child (thread) is destroyed
- passed to the module child_init functions
- destruction happens right after the child_exit functions are called... (which may explain why I think child_exit is redundant and unneeded)

p-trans

- should be a subpool of p-child, but currently is a subpool of permanent_pool, see above
- cleared by the child before going into the accept() loop to receive a connection
- used as connection->pool

r->pool

- for the main request this is a subpool of connection->pool; for subrequests it is a subpool of the parent request's pool.
- exists until the end of the request (*i.e.*, ap_destroy_sub_req, or in child_main after process_request has finished)
- note that r itself is allocated from r->pool; *i.e.*, r->pool is first created and then r is the first thing palloc()d from it

For almost everything folks do, r->pool is the pool to use. But you can see how other lifetimes, such as pchild, are useful to some modules... such as modules that need to open a database connection once per child, and wish to clean it up when the child dies.

You can also see how some bugs have manifested themselves, such as setting connection->user to a value from r->pool -- in this case connection exists for the lifetime of ptrans, which is longer than r->pool (especially if r->pool is a subrequest!). So the correct thing to do is to allocate from connection->pool.

And there was another interesting bug in [mod_include / mod_cgi](#). You'll see in those that they do this test to decide if they should use r->pool or r->main->pool. In this case the resource that they are registering for cleanup is a child process. If it were registered in r->pool, then the code would wait() for the child when the subrequest finishes. With [mod_include](#) this could be any old #include, and the delay can be up to 3 seconds... and happened quite frequently. Instead the subprocess is registered in r->main->pool which causes it to be cleaned up when the entire request is done -- *i.e.*, after the output has been sent to the client and logging has happened.

Tracking open files, etc.

As indicated above, resource pools are also used to track other sorts of resources besides memory. The most common are open files. The

routine which is typically used for this is `ap_pfdopen`, which takes a resource pool and two strings as arguments; the strings are the same as the typical arguments to `fopen`, e.g.,

```
...  
FILE *f = ap_pfdopen (r->pool, r->filename, "r");  
  
if (f == NULL) { ... } else { ... }
```

There is also a `ap_popenf` routine, which parallels the lower-level open system call. Both of these routines arrange for the file to be closed when the resource pool in question is cleared.

Unlike the case for memory, there *are* functions to close files allocated with `ap_pfdopen`, and `ap_popenf`, namely `ap_pfdclose` and `ap_pclosef`. (This is because, on many systems, the number of files which a single process can have open is quite limited). It is important to use these functions to close files allocated with `ap_pfdopen` and `ap_popenf`, since to do otherwise could cause fatal errors on systems such as Linux, which react badly if the same `FILE*` is closed more than once.

(Using the `close` functions is not mandatory, since the file will eventually be closed regardless, but you should consider it in cases where your module is opening, or could open, a lot of files).

Other sorts of resources -- cleanup functions

More text goes here. Describe the cleanup primitives in terms of which the file stuff is implemented; also, `spawn_process`.

Pool cleanups live until `clear_pool()` is called: `clear_pool(a)` recursively calls `destroy_pool()` on all subpools of `a`; then calls all the cleanups for `a`; then releases all the memory for `a`. `destroy_pool(a)` calls `clear_pool(a)` and then releases the pool structure itself. *i.e.*,

`clear_pool(a)` doesn't delete `a`, it just frees up all the resources and you can start using it again immediately.

Fine control -- creating and dealing with sub-pools, with a note on sub-requests

On rare occasions, too-free use of `ap_palloc()` and the associated primitives may result in undesirably profligate resource allocation. You can deal with such a case by creating a *sub-pool*, allocating within the sub-pool rather than the main pool, and clearing or destroying the sub-pool, which releases the resources which were associated with it. (This really *is* a rare situation; the only case in which it comes up in the standard module set is in case of listing directories, and then only with *very* large directories. Unnecessary use of the primitives discussed here can hair up your code quite a bit, with very little gain).

The primitive for creating a sub-pool is `ap_make_sub_pool`, which takes another pool (the parent pool) as an argument. When the main pool is cleared, the sub-pool will be destroyed. The sub-pool may also be cleared or destroyed at any time, by calling the functions `ap_clear_pool` and `ap_destroy_pool`, respectively. (The difference is that `ap_clear_pool` frees resources associated with the pool, while `ap_destroy_pool` also deallocates the pool itself. In the former case, you can allocate new resources within the pool, and clear it again, and so forth; in the latter case, it is simply gone).

One final note -- sub-requests have their own resource pools, which are sub-pools of the resource pool for the main request. The polite way to reclaim the resources associated with a sub request which you have allocated (using the `ap_sub_req_...` functions) is `ap_destroy_sub_req`, which frees the resource pool. Before calling this function, be sure to copy anything that you care about which might be allocated in the sub-request's resource pool into someplace a little less volatile (for instance, the filename in its `request_rec` structure).

(Again, under most circumstances, you shouldn't feel obliged to call this function; only 2K of memory or so are allocated for a typical sub request, and it will be freed anyway when the main request pool is cleared. It is only when you are allocating many, many sub-requests for a single main request that you should seriously consider the `ap_destroy_...` functions).



Configuration, Commands and the Core

One of the design goals for this server was to maintain external compatibility with the NCSA 1.3 server --- that is, to read the same configuration files, to process all the directives therein correctly, and in general to be a drop-in replacement for NCSA. On the other hand, another design goal was to move as much of the server's functionality into modules which have as little as possible to do with the monolithic server core. The only way to reconcile these goals is to move the handling of most commands from the central server into the modules.

However, just giving the modules command tables is not enough to divorce them completely from the server core. The server has to remember the commands in order to act on them later. That involves maintaining data which is private to the modules, and which can be either per-server, or per-directory. Most things are per-directory, including in particular access control and authorization information, but also information on how to determine file types from suffixes, which can be modified by [AddType](#) and [DefaultType](#) directives, and so forth. In general, the governing philosophy is that anything which *can* be made configurable by directory should be; per-server information is generally used in the standard set of modules for information like [Aliases](#) and [Redirects](#) which come into play before the request is tied to a particular place in the underlying file system.

Another requirement for emulating the NCSA server is being able to handle the per-directory configuration files, generally called `.htaccess` files, though even in the NCSA server they can contain directives which have nothing at all to do with access control. Accordingly, after URI -> filename translation, but before performing any other phase, the server walks down the directory hierarchy of the underlying filesystem, following the translated pathname, to read any `.htaccess` files which might be present. The information which is read in then has to be *merged* with the applicable information from the server's own config files (either from the [<Directory>](#) sections in `access.conf`, or from defaults in `srml.conf`,

which actually behaves for most purposes almost exactly like `<Directory />`).

Finally, after having served a request which involved reading `.htaccess` files, we need to discard the storage allocated for handling them. That is solved the same way it is solved wherever else similar problems come up, by tying those structures to the per-transaction resource pool.

Per-directory configuration structures

Let's look out how all of this plays out in `mod_mime.c`, which defines the file typing handler which emulates the NCSA server's behavior of determining file types from suffixes. What we'll be looking at, here, is the code which implements the `AddType` and `AddEncoding` commands. These commands can appear in `.htaccess` files, so they must be handled in the module's private per-directory data, which in fact, consists of two separate tables for MIME types and encoding information, and is declared as follows:

```
typedef struct {
    table *forced_types;      /* Additional AddTyped stuff */
    table *encoding_types;   /* Added with AddEncoding... */
} mime_dir_config;
```

When the server is reading a configuration file, or `<Directory>` section, which includes one of the MIME module's commands, it needs to create a `mime_dir_config` structure, so those commands have something to act on. It does this by invoking the function it finds in the module's ``create per-dir config slot'`, with two arguments: the name of the directory to which this configuration information applies (or `NULL` for `srn.conf`), and a pointer to a resource pool in which the allocation should happen.

(If we are reading a `.htaccess` file, that resource pool is the per-request resource pool for the request; otherwise it is a resource pool which is used for configuration data, and cleared on restarts. Either way, it is

important for the structure being created to vanish when the pool is cleared, by registering a cleanup on the pool if necessary).

For the MIME module, the per-dir config creation function just `ap_pallocs` the structure above, and a creates a couple of tables to fill it. That looks like this:

```
void *create_mime_dir_config (pool *p, char *dummy)
{
    mime_dir_config *new =
        (mime_dir_config *) ap_palloc (p, sizeof(mime_dir_config));

    new->forced_types = ap_make_table (p, 4);
    new->encoding_types = ap_make_table (p, 4);

    return new;
}
```

Now, suppose we've just read in a `.htaccess` file. We already have the per-directory configuration structure for the next directory up in the hierarchy. If the `.htaccess` file we just read in didn't have any [AddType](#) or [AddEncoding](#) commands, its per-directory config structure for the MIME module is still valid, and we can just use it. Otherwise, we need to merge the two structures somehow.

To do that, the server invokes the module's per-directory config merge function, if one is present. That function takes three arguments: the two structures being merged, and a resource pool in which to allocate the result. For the MIME module, all that needs to be done is overlay the tables from the new per-directory config structure with those from the parent:

```
void *merge_mime_dir_configs (pool *p, void *parent_dirv, void
*subdirv)
{
    mime_dir_config *parent_dir = (mime_dir_config *)parent_dirv;
    mime_dir_config *subdir = (mime_dir_config *)subdirv;
    mime_dir_config *new =
        (mime_dir_config *)ap_palloc (p, sizeof(mime_dir_config));
```

```
new->forced_types = ap_overlay_tables (p, subdir->forced_types,
parent_dir->forced_types);
new->encoding_types = ap_overlay_tables (p, subdir->encoding_types,
parent_dir->encoding_types);

return new;
}
```

As a note -- if there is no per-directory merge function present, the server will just use the subdirectory's configuration info, and ignore the parent's. For some modules, that works just fine (e.g., for the includes module, whose per-directory configuration information consists solely of the state of the XBITHACK), and for those modules, you can just not declare one, and leave the corresponding structure slot in the module itself NULL.

Command handling

Now that we have these structures, we need to be able to figure out how to fill them. That involves processing the actual [AddType](#) and [AddEncoding](#) commands. To find commands, the server looks in the module's command table. That table contains information on how many arguments the commands take, and in what formats, where it is permitted, and so forth. That information is sufficient to allow the server to invoke most command-handling functions with pre-parsed arguments. Without further ado, let's look at the [AddType](#) command handler, which looks like this (the [AddEncoding](#) command looks basically the same, and won't be shown here):

```
char *add_type(cmd_parms *cmd, mime_dir_config *m, char *ct, char
*ext)
{
    if (*ext == '.') ++ext;
    ap_table_set (m->forced_types, ext, ct);
    return NULL;
}
```

This command handler is unusually simple. As you can see, it takes four

arguments, two of which are pre-parsed arguments, the third being the per-directory configuration structure for the module in question, and the fourth being a pointer to a `cmd_parms` structure. That structure contains a bunch of arguments which are frequently of use to some, but not all, commands, including a resource pool (from which memory can be allocated, and to which cleanups should be tied), and the (virtual) server being configured, from which the module's per-server configuration data can be obtained if required.

Another way in which this particular command handler is unusually simple is that there are no error conditions which it can encounter. If there were, it could return an error message instead of `NULL`; this causes an error to be printed out on the server's `stderr`, followed by a quick exit, if it is in the main config files; for a `.htaccess` file, the syntax error is logged in the server error log (along with an indication of where it came from), and the request is bounced with a server error response (HTTP error status, code 500).

The MIME module's command table has entries for these commands, which look like this:

```
command_rec mime_cmds[] = {
    { "AddType", add_type, NULL, OR_FILEINFO, TAKE2,
      "a mime type followed by a file extension" },
    { "AddEncoding", add_encoding, NULL, OR_FILEINFO, TAKE2,
      "an encoding (e.g., gzip), followed by a file extension" },
    { NULL }
};
```

The entries in these tables are:

- The name of the command
- The function which handles it
- a (`void *`) pointer, which is passed in the `cmd_parms` structure to the command handler --- this is useful in case many similar commands are handled by the same function.

- A bit mask indicating where the command may appear. There are mask bits corresponding to each `AllowOverride` option, and an additional mask bit, `RSRC_CONF`, indicating that the command may appear in the server's own config files, but *not* in any `.htaccess` file.
- A flag indicating how many arguments the command handler wants pre-parsed, and how they should be passed in. `TAKE2` indicates two pre-parsed arguments. Other options are `TAKE1`, which indicates one pre-parsed argument, `FLAG`, which indicates that the argument should be `On` or `Off`, and is passed in as a boolean flag, `RAW_ARGS`, which causes the server to give the command the raw, unparsed arguments (everything but the command name itself). There is also `ITERATE`, which means that the handler looks the same as `TAKE1`, but that if multiple arguments are present, it should be called multiple times, and finally `ITERATE2`, which indicates that the command handler looks like a `TAKE2`, but if more arguments are present, then it should be called multiple times, holding the first argument constant.
- Finally, we have a string which describes the arguments that should be present. If the arguments in the actual config file are not as required, this string will be used to help give a more specific error message. (You can safely leave this `NULL`).

Finally, having set this all up, we have to use it. This is ultimately done in the module's handlers, specifically for its file-typing handler, which looks more or less like this; note that the per-directory configuration structure is extracted from the `request_rec`'s per-directory configuration vector by using the `ap_get_module_config` function.

```
int find_ct(request_rec *r)
{
    int i;
    char *fn = ap_pstrdup (r->pool, r->filename);
    mime_dir_config *conf = (mime_dir_config *)
        ap_get_module_config(r->per_dir_config, &mime_module);
```

```

char *type;

if (S_ISDIR(r->finfo.st_mode)) {
    r->content_type = DIR_MAGIC_TYPE;
    return OK;
}

if((i=ap_rind(fn, '.')) < 0) return DECLINED;
++i;

if ((type = ap_table_get (conf->encoding_types, &fn[i])))
{
    r->content_encoding = type;

    /* go back to previous extension to try to use it as a type */
    fn[i-1] = '\0';
    if((i=ap_rind(fn, '.')) < 0) return OK;
    ++i;
}

if ((type = ap_table_get (conf->forced_types, &fn[i])))
{
    r->content_type = type;
}

return OK;
}

```

Side notes -- per-server configuration, virtual servers, etc.

The basic ideas behind per-server module configuration are basically the same as those for per-directory configuration; there is a creation function and a merge function, the latter being invoked where a virtual server has partially overridden the base server configuration, and a combined structure must be computed. (As with per-directory configuration, the default if no merge function is specified, and a module is configured in some virtual server, is that the base configuration is simply ignored).

The only substantial difference is that when a command needs to configure the per-server private module data, it needs to go to the `cmd_parms` data to get at it. Here's an example, from the `alias` module, which also indicates how a syntax error can be returned (note that the per-directory configuration argument to the command handler is declared

as a dummy, since the module doesn't actually have per-directory config data):

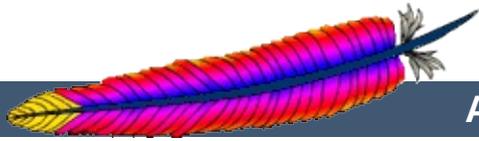
```
char *add_redirect(cmd_parms *cmd, void *dummy, char *f, char *url)
{
    server_rec *s = cmd->server;
    alias_server_conf *conf = (alias_server_conf *)
        ap_get_module_config(s->module_config,&alias_module);
    alias_entry *new = ap_push_array (conf->redirects);

    if (!ap_is_url (url)) return "Redirect to non-URL";

    new->fake = f; new->real = url;
    return NULL;
}
```

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

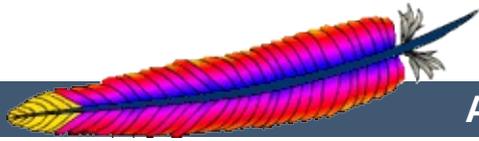
[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

Debugging Memory Allocation in APR

This document has been removed.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

Documenting Apache 2.0

Apache 2.0 uses [Doxygen](#) to document the APIs and global variables in the code. This will explain the basics of how to document using Doxygen.



To start a documentation block, use `/**`

To end a documentation block, use `*/`

In the middle of the block, there are multiple tags we can use:

```
Description of this functions purpose
@param parameter_name description
@return description
@deffunc signature of the function
```

The `deffunc` is not always necessary. Doxygen does not have a full parser in it, so any prototype that use a macro in the return type declaration is too complex for scandoc. Those functions require a `deffunc`. An example (using `>` rather than `>`):

```
/**
 * return the final element of the pathname
 * @param pathname The path to get the final element of
 * @return the final element of the path
 * @tip Examples:
 * <pre>
 * "/foo/bar/gum" -&gt; "gum"
 * "/foo/bar/gum/" -&gt; ""
 * "gum" -&gt; "gum"
 * "wi\n32\stuff" -&gt; "stuff"
 * </pre>
 * @deffunc const char * ap_filename_of_pathname(const char *pathname)
 */
```

At the top of the header file, always include:

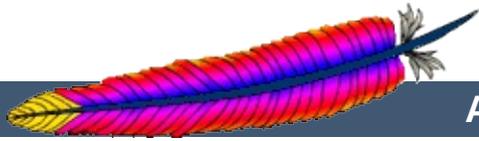
```
/**
 * @package Name of library header
 */
```

Doxygen uses a new HTML file for each package. The HTML files are named `{Name_of_library_header}.html`, so try to be concise with your names.

For a further discussion of the possibilities please refer to [the Doxygen site](#).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

Apache 2.0 Hook Functions

Warning

This document is still in development and may be partially out of date.

In general, a hook function is one that Apache will call at some point during the processing of a request. Modules can provide functions that are called, and specify when they get called in comparison to other modules.



Creating a hook function

In order to create a new hook, four things need to be done:

Declare the hook function

Use the `AP_DECLARE_HOOK` macro, which needs to be given the return type of the hook function, the name of the hook, and the arguments. For example, if the hook returns an `int` and takes a `request_rec *` and an `int` and is called `do_something`, then declare it like this:

```
AP_DECLARE_HOOK(int, do_something, (request_rec *r, int n))
```

This should go in a header which modules will include if they want to use the hook.

Create the hook structure

Each source file that exports a hook has a private structure which is used to record the module functions that use the hook. This is declared as follows:

```
APR_HOOK_STRUCT(  
    APR_HOOK_LINK(do_something)  
    ...  
)
```

Implement the hook caller

The source file that exports the hook has to implement a function that will call the hook. There are currently three possible ways to do this. In all cases, the calling function is called `ap_run_hookname()`.

Void hooks

If the return value of a hook is `void`, then all the hooks are called, and the caller is implemented like this:

```
AP_IMPLEMENT_HOOK_VOID(do_something, (request_rec *r, int n), (r, n))
```

The second and third arguments are the dummy argument declaration and the dummy arguments as they will be used when calling the hook. In other words, this macro expands to something like this:

```
void ap_run_do_something(request_rec *r, int n)
{
    ...
    do_something(r, n);
}
```

Hooks that return a value

If the hook returns a value, then it can either be run until the first hook that does something interesting, like so:

```
AP_IMPLEMENT_HOOK_RUN_FIRST(int, do_something, (request_rec *r, int n), (r, n), DECLINED)
```

The first hook that does *not* return DECLINED stops the loop and its return value is returned from the hook caller. Note that DECLINED is the traditional hook return value meaning "I didn't do anything", but it can be whatever suits you.

Alternatively, all hooks can be run until an error occurs. This boils down to permitting *two* return values, one of which means "I did something, and it was OK" and the other meaning "I did nothing". The first function that returns a value other than one of those two stops the loop, and its return is the return value. Declare these like so:

```
AP_IMPLEMENT_HOOK_RUN_ALL(int, do_something, (request_rec *r, int n), (r, n), OK, DECLINED)
```

Again, OK and DECLINED are the traditional values. You can use what you want.

Call the hook callers

At appropriate moments in the code, call the hook caller, like so:

```
int n, ret;  
request_rec *r;  
  
ret=ap_run_do_something(r, n);
```



A module that wants a hook to be called needs to do two things.

Implement the hook function

Include the appropriate header, and define a static function of the correct type:

```
static int my_something_doer(request_rec *r, int n)
{
    ...
    return OK;
}
```

Add a hook registering function

During initialisation, the server will call each modules hook registering function, which is included in the module structure:

```
static void my_register_hooks()
{
    ap_hook_do_something(my_something_doer, NULL, NULL,
        APR_HOOK_MIDDLE);
}

module MODULE_VAR_EXPORT my_module =
{
    ...
    my_register_hooks /* register hooks */
};
```

Controlling hook calling order

In the example above, we didn't use the three arguments in the hook registration function that control calling order. There are two mechanisms for doing this. The first, rather crude, method, allows us to specify roughly where the hook is run relative to other modules. The final argument control this. There are three possible values: `APR_HOOK_FIRST`, `APR_HOOK_MIDDLE` and `APR_HOOK_LAST`.

All modules using any particular value may be run in any order relative to each other, but, of course, all modules using `APR_HOOK_FIRST` will be run before `APR_HOOK_MIDDLE` which are before `APR_HOOK_LAST`. Modules that don't care when they are run should use `APR_HOOK_MIDDLE`. *These values are spaced out, so that positions like `APR_HOOK_FIRST-2` are possible to hook slightly earlier than other functions.*

Note that there are two more values, `APR_HOOK_REALLY_FIRST` and `APR_HOOK_REALLY_LAST`. These should only be used by the hook exporter.

The other method allows finer control. When a module knows that it must be run before (or after) some other modules, it can specify them by name. The second (third) argument is a NULL-terminated array of strings consisting of the names of modules that must be run before (after) the current module. For example, suppose we want "mod_xyz.c" and "mod_abc.c" to run before we do, then we'd hook as follows:

```
static void register_hooks()
{
    static const char * const aszPre[] = { "mod_xyz.c", "mod_abc.c",
    NULL };

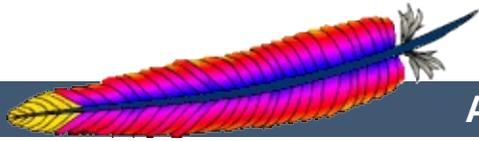
    ap_hook_do_something(my_something_doer, aszPre, NULL,
    APR_HOOK_MIDDLE);
}
```

Note that the sort used to achieve this is stable, so ordering set by `APR_HOOK_ORDER` is preserved, as far as is possible.

Ben Laurie, 15th August 1999

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

Converting Modules from Apache 1.3 to Apache 2.0

This is a first attempt at writing the lessons I learned when trying to convert the `mod_mmap_static` module to Apache 2.0. It's by no means definitive and probably won't even be correct in some ways, but it's a start.



Cleanup Routines

These now need to be of type `apr_status_t` and return a value of that type. Normally the return value will be `APR_SUCCESS` unless there is some need to signal an error in the cleanup. Be aware that even though you signal an error not all code yet checks and acts upon the error.

Initialisation Routines

These should now be renamed to better signify where they sit in the overall process. So the name gets a small change from `mmap_init` to `mmap_post_config`. The arguments passed have undergone a radical change and now look like

- `apr_pool_t *p`
- `apr_pool_t *plog`
- `apr_pool_t *ptemp`
- `server_rec *s`

Data Types

A lot of the data types have been moved into the [APR](#). This means that some have had a name change, such as the one shown above. The following is a brief list of some of the changes that you are likely to have to make.

- `pool` becomes `apr_pool_t`
- `table` becomes `apr_table_t`



Register Hooks

The new architecture uses a series of hooks to provide for calling your functions. These you'll need to add to your module by way of a new function, `static void register_hooks(void)`. The function is really reasonably straightforward once you understand what needs to be done. Each function that needs calling at some stage in the processing of a request needs to be registered, handlers do not. There are a number of phases where functions can be added, and for each you can specify with a high degree of control the relative order that the function will be called in.

This is the code that was added to `mod_mmap_static`:

```
static void register_hooks(void)
{
    static const char * const aszPre[]={ "http_core.c",NULL };
    ap_hook_post_config(mmap_post_config,NULL,NULL,HOOK_MIDDLE);
    ap_hook_translate_name(mmap_static_xlat,aszPre,NULL,HOOK_LAST);
};
```

This registers 2 functions that need to be called, one in the `post_config` stage (virtually every module will need this one) and one for the `translate_name` phase. note that while there are different function names the format of each is identical. So what is the format?

```
ap_hook_phase_name(function_name, predecessors, successors, position);
```

There are 3 hook positions defined...

- `HOOK_FIRST`
- `HOOK_MIDDLE`
- `HOOK_LAST`

To define the position you use the position and then modify it with the

predecessors and successors. Each of the modifiers can be a list of functions that should be called, either before the function is run (predecessors) or after the function has run (successors).

In the `mod_mmap_static` case I didn't care about the `post_config` stage, but the `mmap_static_xlat` **must** be called after the core module had done its name translation, hence the use of the `aszPre` to define a modifier to the position `HOOK_LAST`.

Module Definition

There are now a lot fewer stages to worry about when creating your module definition. The old definition looked like

```
module MODULE_VAR_EXPORT module_name_module =
{
    STANDARD_MODULE_STUFF,
    /* initializer */
    /* dir config creator */
    /* dir merger --- default is to override */
    /* server config */
    /* merge server config */
    /* command handlers */
    /* handlers */
    /* filename translation */
    /* check_user_id */
    /* check auth */
    /* check access */
    /* type_checker */
    /* fixups */
    /* logger */
    /* header parser */
    /* child_init */
    /* child_exit */
    /* post read-request */
};
```

The new structure is a great deal simpler...

```
module MODULE_VAR_EXPORT module_name_module =
{
    STANDARD20_MODULE_STUFF,
```

```
/* create per-directory config structures */
/* merge per-directory config structures */
/* create per-server config structures */
/* merge per-server config structures */
/* command handlers */
/* handlers */
/* register hooks */
};
```

Some of these read directly across, some don't. I'll try to summarise what should be done below.

The stages that read directly across :

```
/* dir config creator */
    /* create per-directory config structures */
/* server config */
    /* create per-server config structures */
/* dir merger */
    /* merge per-directory config structures */
/* merge server config */
    /* merge per-server config structures */
/* command table */
    /* command apr_table_t */
/* handlers */
    /* handlers */
```

The remainder of the old functions should be registered as hooks. There are the following hook stages defined so far...

ap_hook_post_config

this is where the old `_init` routines get registered

ap_hook_http_method

retrieve the http method from a request. (legacy)

ap_hook_open_logs

open any specified logs

ap_hook_auth_checker

check if the resource requires authorization

ap_hook_access_checker

check for module-specific restrictions

ap_hook_check_user_id

check the user-id and password

ap_hook_default_port

retrieve the default port for the server

ap_hook_pre_connection

do any setup required just before processing, but after accepting

ap_hook_process_connection

run the correct protocol

ap_hook_child_init

call as soon as the child is started

ap_hook_create_request

??

ap_hook_fixups

last chance to modify things before generating content

ap_hook_handler

generate the content

ap_hook_header_parser

lets modules look at the headers, not used by most modules, because they use `post_read_request` for this

ap_hook_insert_filter

to insert filters into the filter chain

ap_hook_log_transaction

log information about the request

ap_hook_optional_fn_retrieve

retrieve any functions registered as optional

ap_hook_post_read_request

called after reading the request, before any other phase

ap_hook_quick_handler

called before any request processing, used by cache modules.

ap_hook_translate_name

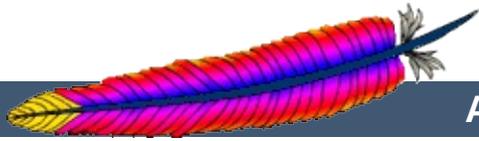
translate the URI into a filename

ap_hook_type_checker

determine and/or set the doc type

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

Request Processing in the Apache HTTP Server 2.x

Warning

Warning - this is a first (fast) draft that needs further revision!

Several changes in 2.0 and above affect the internal request processing mechanics. Module authors need to be aware of these changes so they may take advantage of the optimizations and security enhancements.

The first major change is to the subrequest and redirect mechanisms. There were a number of different code paths in the Apache HTTP Server 1.3 to attempt to optimize subrequest or redirect behavior. As patches were introduced to 2.0, these optimizations (and the server behavior) were quickly broken due to this duplication of code. All duplicate code has been folded back into `ap_process_request_internal()` to prevent the code from falling out of sync again.

This means that much of the existing code was 'unoptimized'. It is the Apache HTTP Project's first goal to create a robust and correct implementation of the HTTP server RFC. Additional goals include security, scalability and optimization. New methods were sought to optimize the server (beyond the performance of 1.3) without introducing fragile or insecure code.



The Request Processing Cycle

All requests pass through `ap_process_request_internal()` in `request.c`, including subrequests and redirects. If a module doesn't pass generated requests through this code, the author is cautioned that the module may be broken by future changes to request processing.

To streamline requests, the module author can take advantage of the hooks offered to drop out of the request cycle early, or to bypass core hooks which are irrelevant (and costly in terms of CPU.)



Unescapes the URL

The request's `parsed_uri` path is unescaped, once and only once, at the beginning of internal request processing.

This step is bypassed if the `proxyreq` flag is set, or the `parsed_uri.path` element is unset. The module has no further control of this one-time unescape operation, either failing to unescape or multiply unescaping the URL leads to security repercussions.

Strips Parent and This Elements from the URI

All `/../` and `/./` elements are removed by `ap_getparents()`. This helps to ensure the path is (nearly) absolute before the request processing continues.

This step cannot be bypassed.

Initial URI Location Walk

Every request is subject to an `ap_location_walk()` call. This ensures that `<Location>` sections are consistently enforced for all requests. If the request is an internal redirect or a sub-request, it may borrow some or all of the processing from the previous or parent request's `ap_location_walk`, so this step is generally very efficient after processing the main request.

translate_name

Modules can determine the file name, or alter the given URI in this step. For example, `mod_vhost_alias` will translate the URI's path into the configured virtual host, `mod_alias` will translate the path to an alias path, and if the request falls back on the core, the `DocumentRoot` is prepended to the request resource.

If all modules `DECLINE` this phase, an error 500 is returned to the browser, and a "couldn't translate name" error is logged automatically.

Hook: `map_to_storage`

After the file or correct URI was determined, the appropriate per-dir configurations are merged together. For example, `mod_proxy` compares and merges the appropriate `<Proxy>` sections. If the URI is nothing more than a local (non-proxy) TRACE request, the core handles the request and returns `DONE`. If no module answers this hook with `OK` or `DONE`, the core will run the request filename against the `<Directory>` and `<Files>` sections. If the request 'filename' isn't an absolute, legal filename, a note is set for later termination.

URI Location Walk

Every request is hardened by a second `ap_location_walk()` call. This reassures that a translated request is still subjected to the configured `<Location>` sections. The request again borrows some or all of the processing from its previous `location_walk` above, so this step is almost always very efficient unless the translated URI mapped to a substantially different path or Virtual Host.

Hook: `header_parser`

The main request then parses the client's headers. This prepares the remaining request processing steps to better serve the client's request.



Needs Documentation. Code is:

```
switch (ap_satisfies(r)) {
case SATISFY_ALL:
case SATISFY_NOSPEC:
    if ((access_status = ap_run_access_checker(r)) != 0) {
        return decl_die(access_status, "check access", r);
    }

    if (ap_some_auth_required(r)) {
        if (((access_status = ap_run_check_user_id(r)) != 0)
            || !ap_auth_type(r)) {
            return decl_die(access_status, ap_auth_type(r)
                ? "check user. No user file?"
                : "perform authentication. AuthType not set!"
                r);
        }

        if (((access_status = ap_run_auth_checker(r)) != 0)
            || !ap_auth_type(r)) {
            return decl_die(access_status, ap_auth_type(r)
                ? "check access. No groups file?"
                : "perform authentication. AuthType not set!"
                r);
        }
    }
    break;

case SATISFY_ANY:
    if (((access_status = ap_run_access_checker(r)) != 0) {
        if (!ap_some_auth_required(r)) {
            return decl_die(access_status, "check access", r);
        }
    }

    if (((access_status = ap_run_check_user_id(r)) != 0)
        || !ap_auth_type(r)) {
        return decl_die(access_status, ap_auth_type(r)
            ? "check user. No user file?"
            : "perform authentication. AuthType not set!"
            r);
    }

    if (((access_status = ap_run_auth_checker(r)) != 0)
        || !ap_auth_type(r)) {
        return decl_die(access_status, ap_auth_type(r)
            ? "check access. No groups file?"
            : "perform authentication. AuthType not set!"
```

```
        r);  
    }  
    }  
    break;  
}
```



Hook: `type_checker`

The modules have an opportunity to test the URI or filename against the target resource, and set mime information for the request. Both `mod_mime` and `mod_mime_magic` use this phase to compare the file name or contents against the administrator's configuration and set the content type, language, character set and request handler. Some modules may set up their filters or other request handling parameters at this time.

If all modules `DECLINE` this phase, an error 500 is returned to the browser, and a "couldn't find types" error is logged automatically.

Hook: `fixups`

Many modules are 'trounced' by some phase above. The `fixups` phase is used by modules to 'reassert' their ownership or force the request's fields to their appropriate values. It isn't always the cleanest mechanism, but occasionally it's the only option.



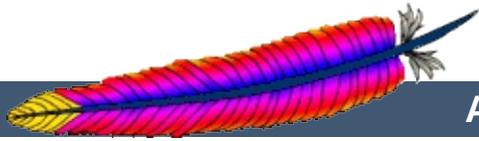
This phase is **not** part of the processing in `ap_process_request_internal()`. Many modules prepare one or more subrequests prior to creating any content at all. After the core, or a module calls `ap_process_request_internal()` it then calls `ap_invoke_handler()` to generate the request.

Hook: insert_filter

Modules that transform the content in some way can insert their values and override existing filters, such that if the user configured a more advanced filter out-of-order, then the module can move its order as needed. There is no result code, so actions in this hook better be trusted to always succeed.

Hook: handler

The module finally has a chance to serve the request in its handler hook. Note that not every prepared request is sent to the handler hook. Many modules, such as `mod_autoindex`, will create subrequests for a given URI, and then never serve the subrequest, but simply lists it for the user. Remember not to put required teardown from the hooks above into this module, but register pool cleanups against the request pool to free resources as required.



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

How filters work in Apache 2.0

Warning

This is a cut 'n paste job from an email (<022501c1c529\$63a9550\$7f00000a@KOJ>) and only reformatted for better readability. It's not up to date but may be a good start for further research.



Filter types

There are three basic filter types (each of these is actually broken down into two categories, but that comes later).

CONNECTION

Filters of this type are valid for the lifetime of this connection. (AP_FTYPE_CONNECTION, AP_FTYPE_NETWORK)

PROTOCOL

Filters of this type are valid for the lifetime of this request from the point of view of the client, this means that the request is valid from the time that the request is sent until the time that the response is received. (AP_FTYPE_PROTOCOL, AP_FTYPE_TRANSCODE)

RESOURCE

Filters of this type are valid for the time that this content is used to satisfy a request. For simple requests, this is identical to PROTOCOL, but internal redirects and sub-requests can change the content without ending the request. (AP_FTYPE_RESOURCE, AP_FTYPE_CONTENT_SET)

It is important to make the distinction between a protocol and a resource filter. A resource filter is tied to a specific resource, it may also be tied to header information, but the main binding is to a resource. If you are writing a filter and you want to know if it is resource or protocol, the correct question to ask is: "Can this filter be removed if the request is redirected to a different resource?" If the answer is yes, then it is a resource filter. If it is no, then it is most likely a protocol or connection filter. I won't go into connection filters, because they seem to be well understood. With this definition, a few examples might help:

Byterange

We have coded it to be inserted for all requests, and it is removed if not used. Because this filter is active at the beginning of all requests, it can not be removed if it is redirected, so this is a protocol filter.

http_header

This filter actually writes the headers to the network. This is obviously a required filter (except in the asis case which is special and will be dealt with below) and so it is a protocol filter.

Deflate

The administrator configures this filter based on which file has been requested. If we do an internal redirect from an autoindex page to an index.html page, the deflate filter may be added or removed based on config, so this is a resource filter.

The further breakdown of each category into two more filter types is strictly for ordering. We could remove it, and only allow for one filter type, but the order would tend to be wrong, and we would need to hack things to make it work. Currently, the RESOURCE filters only have one filter type, but that should change.



This is actually rather simple in theory, but the code is complex. First of all, it is important that everybody realize that there are three filter lists for each request, but they are all concatenated together. So, the first list is `r->output_filters`, then `r->proto_output_filters`, and finally `r->connection->output_filters`. These correspond to the RESOURCE, PROTOCOL, and CONNECTION filters respectively. The problem previously, was that we used a singly linked list to create the filter stack, and we started from the "correct" location. This means that if I had a RESOURCE filter on the stack, and I added a CONNECTION filter, the CONNECTION filter would be ignored. This should make sense, because we would insert the connection filter at the top of the `c->output_filters` list, but the end of `r->output_filters` pointed to the filter that used to be at the front of `c->output_filters`. This is obviously wrong. The new insertion code uses a doubly linked list. This has the advantage that we never lose a filter that has been inserted. Unfortunately, it comes with a separate set of headaches.

The problem is that we have two different cases where we use subrequests. The first is to insert more data into a response. The second is to replace the existing response with an internal redirect. These are two different cases and need to be treated as such.

In the first case, we are creating the subrequest from within a handler or filter. This means that the next filter should be passed to `make_sub_request` function, and the last resource filter in the sub-request will point to the next filter in the main request. This makes sense, because the sub-request's data needs to flow through the same set of filters as the main request. A graphical representation might help:

```
Default_handler --> includes_filter --> byterange --> ...
```

If the includes filter creates a sub request, then we don't want the data from that sub-request to go through the includes filter, because it might

not be SSI data. So, the subrequest adds the following:

```
Default_handler --> includes_filter -/-> byterange --> ...  
/   
Default_handler --> sub_request_core
```

What happens if the subrequest is SSI data? Well, that's easy, the `includes_filter` is a resource filter, so it will be added to the sub request in between the `Default_handler` and the `sub_request_core` filter.

The second case for sub-requests is when one sub-request is going to become the real request. This happens whenever a sub-request is created outside of a handler or filter, and NULL is passed as the next filter to the `make_sub_request` function.

In this case, the resource filters no longer make sense for the new request, because the resource has changed. So, instead of starting from scratch, we simply point the front of the resource filters for the sub-request to the front of the protocol filters for the old request. This means that we won't lose any of the protocol filters, neither will we try to send this data through a filter that shouldn't see it.

The problem is that we are using a doubly-linked list for our filter stacks now. But, you should notice that it is possible for two lists to intersect in this model. So, you do you handle the previous pointer? This is a very difficult question to answer, because there is no "right" answer, either method is equally valid. I looked at why we use the previous pointer. The only reason for it is to allow for easier addition of new servers. With that being said, the solution I chose was to make the previous pointer always stay on the original request.

This causes some more complex logic, but it works for all cases. My concern in having it move to the sub-request, is that for the more common case (where a sub-request is used to add data to a response),

the main filter chain would be wrong. That didn't seem like a good idea to me.



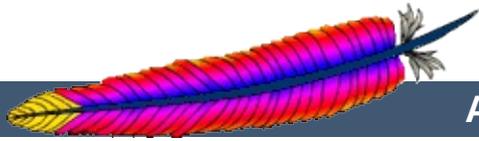
The final topic. :-) Mod_Asis is a bit of a hack, but the handler needs to remove all filters except for connection filters, and send the data. If you are using mod_asis, all other bets are off.



The absolutely last point is that the reason this code was so hard to get right, was because we had hacked so much to force it to work. I wrote most of the hacks originally, so I am very much to blame. However, now that the code is right, I have started to remove some hacks. Most people should have seen that the `reset_filters` and `add_required_filters` functions are gone. Those inserted protocol level filters for error conditions, in fact, both functions did the same thing, one after the other, it was really strange. Because we don't lose protocol filters for error cases any more, those hacks went away. The `HTTP_HEADER`, `Content-length`, and `byterange` filters are all added in the `insert_filters` phase, because if they were added earlier, we had some interesting interactions. Now, those could all be moved to be inserted with the `HTTP_IN`, `CORE`, and `CORE_IN` filters. That would make the code easier to follow.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Developer Documentation](#)

Apache 2.0 Thread Safety Issues

When using any of the threaded mpms in Apache 2.0 it is important that every function called from Apache be thread safe. When linking in 3rd party extensions it can be difficult to determine whether the resulting server will be thread safe. Casual testing generally won't tell you this either as thread safety problems can lead to subtle race conditons that may only show up in certain conditions under heavy load.



Global and Static Variables

When writing your module or when trying to determine if a module or 3rd party library is thread safe there are some common things to keep in mind.

First, you need to recognize that in a threaded model each individual thread has its own program counter, stack and registers. Local variables live on the stack, so those are fine. You need to watch out for any static or global variables. This doesn't mean that you are absolutely not allowed to use static or global variables. There are times when you actually want something to affect all threads, but generally you need to avoid using them if you want your code to be thread safe.

In the case where you have a global variable that needs to be global and accessed by all threads, be very careful when you update it. If, for example, it is an incrementing counter, you need to atomically increment it to avoid race conditions with other threads. You do this using a mutex (mutual exclusion). Lock the mutex, read the current value, increment it and write it back and then unlock the mutex. Any other thread that wants to modify the value has to first check the mutex and block until it is cleared.

If you are using [APR](#), have a look at the `apr_atomic_*` functions and the `apr_thread_mutex_*` functions.



This is a common global variable that holds the error number of the last error that occurred. If one thread calls a low-level function that sets `errno` and then another thread checks it, we are bleeding error numbers from one thread into another. To solve this, make sure your module or library defines `_REENTRANT` or is compiled with `-D_REENTRANT`. This will make `errno` a per-thread variable and should hopefully be transparent to the code. It does this by doing something like this:

```
#define errno (*(__errno_location()))
```

which means that accessing `errno` will call `__errno_location()` which is provided by the `libc`. Setting `_REENTRANT` also forces redefinition of some other functions to their `*_r` equivalents and sometimes changes the common `getc/putc` macros into safer function calls. Check your `libc` documentation for specifics. Instead of, or in addition to `_REENTRANT` the symbols that may affect this are `_POSIX_C_SOURCE`, `_THREAD_SAFE`, `_SVID_SOURCE`, and `_BSD_SOURCE`.



Not only do things have to be thread safe, but they also have to be reentrant. `strtok()` is an obvious one. You call it the first time with your delimiter which it then remembers and on each subsequent call it returns the next token. Obviously if multiple threads are calling it you will have a problem. Most systems have a reentrant version of the function called `strtok_r()` where you pass in an extra argument which contains an allocated `char *` which the function will use instead of its own static storage for maintaining the tokenizing state. If you are using [APR](#) you can use `apr_strtok()`.

`crypt()` is another function that tends to not be reentrant, so if you run across calls to that function in a library, watch out. On some systems it is reentrant though, so it is not always a problem. If your system has `crypt_r()` chances are you should be using that, or if possible simply avoid the whole mess by using md5 instead.



The following is a list of common libraries that are used by 3rd party Apache modules. You can check to see if your module is using a potentially unsafe library by using tools such as `ldd(1)` and `nm(1)`. For [PHP](#), for example, try this:

```
% ldd libphp4.so
libsablot.so.0 => /usr/local/lib/libsablot.so.0 (0x401f6000)
libexpat.so.0 => /usr/lib/libexpat.so.0 (0x402da000)
libsnpmp.so.0 => /usr/lib/libsnmp.so.0 (0x402f9000)
libpdf.so.1 => /usr/local/lib/libpdf.so.1 (0x40353000)
libz.so.1 => /usr/lib/libz.so.1 (0x403e2000)
libpng.so.2 => /usr/lib/libpng.so.2 (0x403f0000)
libmysqlclient.so.11 => /usr/lib/libmysqlclient.so.11 (0x40411000)
libming.so => /usr/lib/libming.so (0x40449000)
libm.so.6 => /lib/libm.so.6 (0x40487000)
libfreetype.so.6 => /usr/lib/libfreetype.so.6 (0x404a8000)
libjpeg.so.62 => /usr/lib/libjpeg.so.62 (0x404e7000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x40505000)
libssl.so.2 => /lib/libssl.so.2 (0x40532000)
libcrypto.so.2 => /lib/libcrypto.so.2 (0x40560000)
libresolv.so.2 => /lib/libresolv.so.2 (0x40624000)
libdl.so.2 => /lib/libdl.so.2 (0x40634000)
libnsl.so.1 => /lib/libnsl.so.1 (0x40637000)
libc.so.6 => /lib/libc.so.6 (0x4064b000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x80000000)
```

In addition to these libraries you will need to have a look at any libraries linked statically into the module. You can use `nm(1)` to look for individual symbols in the module.



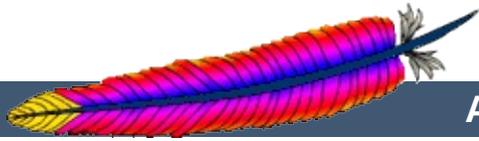
Please drop a note to dev@httpd.apache.org if you have additions or corrections to this list.

Library	Version	Thread Safe?	Notes
ASpell/PSpell		?	
Berkeley DB	3.x, 4.x	Yes	Be careful about sharing a connection a
bzip2		Yes	Both low-level and high-level APIs are t However, high-level API requires thread errno.
cdb		?	
C-Client		Perhaps	c-client uses strtok() and gethostb are not thread-safe on most C library im client's static data is meant to be share If strtok() and gethostbyname() a your OS, c-client <i>may</i> be thread-safe.
libcrypt		?	
Expat		Yes	Need a separate parser instance per th
FreeTDS		?	
FreeType		?	
GD 1.8.x		?	
GD 2.0.x		?	
gdbm		No	Errors returned via a static gdbm_erro
ImageMagick	5.2.2	Yes	ImageMagick docs claim it is thread saf 5.2.2 (see Change log).
Imlib2		?	
libjpeg	v6b	?	
libmysqlclient		Yes	Use mysqlclient_r library variant to ensu For more information, please read http://dev.mysql.com/doc/mysql/en/Thre

Ming	0.2a	?	
Net-SNMP	5.0.x	?	
OpenLDAP	2.1.x	Yes	Use ldap_r library variant to ensure th
OpenSSL	0.9.6g	Yes	Requires proper usage of CRYPTO_num CRYPTO_set_locking_callback, CRYPTO_set_id_callback
liboci8 (Oracle 8+)	8.x,9.x	?	
pdflib	5.0.x	Yes	PDFLib docs claim it is thread safe; cha it has been partially thread-safe since V http://www.pdflib.com/products/pdflib-fa
libpng	1.0.x	?	
libpng	1.2.x	?	
libpq (PostgreSQL)	8.x	Yes	Don't share connections across threads crypt () calls
Sablotron	0.95	?	
zlib	1.1.4	Yes	Relies upon thread-safe zalloc and zfre is to use libc's calloc/free which are thre

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Glossar

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Dieses Glossar erläutert einige gebräuchliche Fachbegriffe im Zusammenhang mit dem Apache im Speziellen und Web-Diensten im Allgemeinen. Weitere Informationen zum jeweiligen Begriff erreichen Sie über die Links.



Algorithmus

Eine eindeutige Formel oder ein Satz von Regeln zur Lösung eines Problems in einer endlichen Anzahl von Schritten. Algorithmen zur Verschlüsselung werden üblicherweise [Chiffre](#) genannt.

APache eXtension Tool (apxs)

Ein Perl-Skript zur Kompilierung von [Modul](#)-Quelltexten zu Dynamic-Shared-Objects ([DSOs](#)) und zur Installation dieser zum Apache-Webserver.

Siehe: [apxs](#)-Dokumentation

Apache Portable Runtime (APR)

Eine Menge von Bibliotheken, die viele grundlegende Schnittstellen zwischen dem Server und dem Betriebssystem anbietet. Die APR wird parallel zum Apache HTTP Server in einem eigenständigen Projekt entwickelt.

Siehe auch: [Apache Portable Runtime Project](#)

Authentifizierung

Die positive Identifizierung einer Netzwerkeinheit, wie z.B. eines Servers, eines Clients oder eines Benutzers.

Siehe: [Authentisierung, Autorisierung und Zugriffskontrolle](#)

Certification Authority [sə'tifi'keɪʃən ɔ:θɔ:riti] (CA)

(Anm.d.Ü.: die Zertifizierungsstelle) Eine vertrauenswürdige dritte Partei, deren Zweck es ist, Zertifikate für Netzwerkeinheiten zu signieren. Andere Netzwerkeinheiten können die Signatur prüfen, um sicherzustellen, dass eine CA den Inhaber eines Zertifikats authentifiziert hat.

Siehe: [SSL/TLS-Verschlüsselung](#)

Certificate Signing Request [sə'tifikit saɪnɪŋ ri'kwɛst] (CSR)

(Anm.d.Ü.: Zertifikats-Signierungsanfrage) Ein unsigniertes [Zertifikat](#) zur Einreichung bei einer [Zertifizierungsstelle](#), welche es mit dem [privaten Schlüssel](#) ihres CA-Zertifikats signiert. Durch die Signatur wird ein CSR zum echten Zertifikat.

Siehe: [SSL/TLS-Verschlüsselung](#)

Chiffre

Die *Chiffre* ist ein Algorithmus oder System zur Datenverschlüsselung. Beispiele sind DES, IDEA, RC4 usw. Im Englischen spricht man von *Cipher* ['saifə]

Siehe: [SSL/TLS-Verschlüsselung](#)

Chiffretext

Das Ergebnis, nachdem ein [Klartext](#) eine [Chiffre](#) durchlaufen hat.

Siehe: [SSL/TLS-Verschlüsselung](#)

Common Gateway Interface ['kɔmən geitwei 'intə:feis] (CGI)

Eine einheitliche Definition einer Schnittstelle zwischen einem Webserver und einem externen Programm, welcher dem externen Programm die Behandlung von Anfragen ermöglicht. Die Schnittstelle ist ursprünglich von der [NCSA](#) definiert worden. Es existiert jedoch auch ein [RFC-Projekt](#).

Siehe: [Dynamische Inhalte mit CGI](#)

CONNECT [kənekt]

Eine [HTTP-Methode](#) zur Weiterleitung von Rohdaten über HTTP. Sie kann dazu verwendet werden, andere Protokolle wie zum Beispiel das SSL-Protokoll zu kapseln.

Digitale Signatur

Ein chiffrierter Textblock, der die Gültigkeit eines Zertifikats oder einer anderen Datei bestätigt. Eine [Zertifizierungsstelle](#) erstellt eine digitale Signatur durch Generierung eines [Hashs](#) aus dem in einem *Zertifikat* enthaltenen *öffentlichen Schlüssel* und anschließender Codierung des Hashs mit dem *privaten Schlüssel* des Zertifikats. Nur der öffentliche Schlüssel der CA kann die Signatur decodieren. So wird sichergestellt, dass die CA die Netwerkeinheit, welche das *Zertifikat* besitzt, authentifiziert hat.

Siehe: [SSL/TLS-Verschlüsselung](#)

Direktive

Eine Konfigurationsanweisung, die das Verhalten des Apache in

einem oder mehreren Punkten steuert. Direktiven werden in den [Konfigurationsdateien](#) gesetzt.

Siehe: [Verzeichnis der Direktiven](#)

Dynamic Shared Object [dai'næmik ʃæd'ɔbdʒikt] **(DSO)**

Separat von der Apache-Binärdatei [httpd](#) kompilierte [Module](#), die bei Bedarf geladen werden können.

Siehe: [Unterstützung für Dynamic-Shared-Objects](#)

exportbeschränkt

Verminderte kryptografische Stärke (und Sicherheit), um den Exportbestimmungen der Vereinigten Staaten (*Anm.d.Ü.:* konkret: United States' Export Administration Regulations (EAR)) zu entsprechen. Exportbeschränkte Verschlüsselungssoftware ist auf eine kurze Schlüssellänge begrenzt, was zu *Chiffretexten* führt, die gewöhnlich mittels Brute-Force dekodiert werden können.

Siehe: [SSL/TLS-Verschlüsselung](#)

Filter

Ein Verfahren, dass auf vom Server empfangene oder zu sendende Daten angewendet wird. Eingabefilter verarbeiten vom Client an den Server gesendete Daten, während Ausgabefilter vom Server an den Client zu sendende Daten verarbeiten. Der Ausgabefilter INCLUDES beispielsweise untersucht Dokumente nach [Server-Side-Includes](#) und führt sie aus.

Siehe: [Filter](#)

Handler ['hændlə]

Eine Apache-interne Darstellung der Aktion, die beim Aufruf einer Datei auszuführen ist. Im Allgemeinen besitzen Dateien implizite, auf dem Dateityp basierende Handler. Gewöhnlich werden alle Dateien vom Server bedient, einige Dateitypen werden jedoch separat "behandelt" (*Anm.d.Ü.:* besitzen einen separaten Handler). Der `cgi-script`-Handler beispielsweise kennzeichnet Dateien, die als [CGI-Programme](#) ausgeführt werden sollen.

Siehe: [Verwendung von Apache-Handlern](#)

Hash [hæʃ]

Ein mathematischer, unumkehrbarer Einweg-Algorithmus zur Generierung einer Zeichenfolge fester Länge aus einer anderen Zeichenfolge beliebiger Länge. Unterschiedliche Zeichenfolgen bei der Eingabe ergeben üblicherweise unterschiedliche Hashes (abhängig von der Hash-Funktion).

Header [hedə]

Der Teil der [HTTP](#)-Anfrage und -Antwort, der vor den eigentlichen Daten übermittelt wird und den Inhalt beschreibende Meta-Informationen enthält.

.htaccess

Eine [Konfigurationsdatei](#), die innerhalb des Web-Verzeichnisbaums abgelegt wird und zu dem Verzeichnis, in dem sie abgelegt ist, sowie allen Unterverzeichnissen [Konfigurationsdirektiven](#) enthält. Trotz ihres Namens kann diese Datei nahezu alle Arten von Direktiven enthalten, nicht nur Direktiven zur Zugriffskontrolle.

Siehe: [Konfigurationsdateien](#)

httpd.conf

Die [Haupt-Konfigurationsdatei](#) ist `/usr/local/apache2/conf/httpd.conf`. Dies kann aber zur Laufzeit oder zur Kompilierungszeit anders konfiguriert werden.

Siehe: [Konfigurationsdateien](#)

HTTPS

Das HyperText-Transfer-Protokoll (Secure), der Standard-Verschlüsselungsmechanismus im World Wide Web. Tatsächlich handelt es sich hierbei um HTTP über [SSL](#).

Siehe: [SSL/TLS-Verschlüsselung](#)

HyperText-Transfer-Protokoll (HTTP)

Das Standard-Übertragungsprotokoll im World Wide Web. Der Apache implementiert die Protokollversion 1.1, bezeichnet als HTTP/1.1 und definiert in [RFC 2616](#).

Klartext

Der unverschlüsselte Text.

Konfigurationsanweisung

Siehe: [Direktive](#)

Konfigurationsdatei

Eine Textdatei mit [Direktiven](#), welche die Konfiguration des Apache steuern.

Siehe: [Konfigurationsdateien](#)

Kontext

Ein Bereich in den [Konfigurationsdateien](#), in dem verschiedene Typen von [Direktiven](#) erlaubt sind.

Siehe: [Erklärung der Fachbegriffe zu Apache-Direktiven](#)

Message-Digest [`'mesidʒ`]

Ein Hash einer Nachricht, mit dem sich sicherstellen läßt, dass der Inhalt der Nachricht während der Übertragung nicht verändert wurde.

(Anm.d.Ü.: ein so genannter Extrakt der Nachricht)

Siehe: [SSL/TLS-Verschlüsselung](#)

Methode

Im [HTTP](#)-Kontext eine in der Anfrage(zeile) des Clients angegeben Aktion, die auf eine Ressource angewendet wird. GET, POST und PUT sind einige der verfügbaren HTTP-Methoden.

MIME-Typ [`maim ty:p`]

Eine Art und Weise, den Typ des übermittelten Dokuments zu beschreiben. Sein Name leitet sich davon ab, dass sein Format den Multipurpose Internet Mail Extensions entlehnt wurde. Er besteht aus einem Haupttyp und einem Untertyp, getrennt durch einen Schrägstrich. Einige Beispiele sind `text/html`, `image/gif` und `application/octet-stream`. Bei HTTP wird der MIME-Typ mit dem [Header](#) Content-Type übermittelt.

Siehe: [mod_mime](#)

Modul

Ein selbstständiger Teil eines Programms. Ein Großteil der

Funktionalität des Apache ist in Modulen enthalten, die Sie einbinden oder entfernen können. In die Apache-Binärdatei [httpd](#) einkompilierte Module werden *statische Module* genannt, während Module, die separat gespeichert sind und optional zur Laufzeit geladen werden können, *dynamische Module* oder [DSOs](#) genannt werden. Standardmäßig eingebundene Module werden *Basismodule* genannt. Für den Apache sind viele Module verfügbar, die nicht als Bestandteil des [Apache-HTTP-Server-Tarballs](#) ausgeliefert werden. Diese werden als *Drittmodule* bezeichnet.

Siehe: [Modulverzeichnis](#)

Module-Magic-Number [ˈmɔju:l mədʒɪk ˈnʌmbə] (MMN)

Die Module-Magic-Number ist eine Konstante, die im Apache-Quelltext definiert ist und im Zusammenhang mit der Binärkompatibilität von Modulen steht. Sie wird geändert, wenn sich interne Apache-Strukturen, -Funktionen oder andere signifikante Teile der API derart ändern, dass eine Binärkompatibilität nicht mehr gewährleistet werden kann. Bei einer MMN-Änderung müssen alle Module von Drittanbietern zumindest neu kompiliert und zuweilen auch geringfügig angepaßt werden, um mit der neuen Apache-Version zu funktionieren.

Öffentlicher Schlüssel

Der öffentlich verfügbare Schlüssel in einem [Public-Key-Kryptographie](#)-System, mit dem für seinen Eigentümer bestimmte Nachrichten verschlüsselt und Signaturen von seinem Eigentümer entschlüsselt werden.

Siehe: [SSL/TLS-Verschlüsselung](#)

OpenSSL [ˈəʊpənɛsɛs ˈɛl]

Das Open-Source-Toolkit für SSL/TLS

Siehe: <http://www.openssl.org/>

Passphrase [pa:freiz]

Das Wort oder die Phrase, welches private Schlüssel-Dateien schützt. Sie verhindert die Entschlüsselung durch nicht autorisierte Benutzer. Normalerweise ist dies einfach der geheimen (De-

)Codierungsschlüssel, der für [Chiffren](#) verwendet wird.

Siehe: [SSL/TLS-Verschlüsselung](#)

Privater Schlüssel

Der geheime Schlüssel in einem [Public-Key-Kryptographie](#)-System, mit dem hereinkommende Nachrichten decodiert und ausgehende signiert werden.

Siehe: [SSL/TLS-Verschlüsselung](#)

Proxy

Ein zwischen dem Client und dem *ursprünglichen Server* (*Anm.d.Ü.: der Server, den der Client tatsächlich erreichen möchte*) liegender Server. Er nimmt Anfragen von Clients entgegen, übermittelt diese Anfragen dem ursprünglichen Server und liefert die Antwort des ursprünglichen Servers an den Client zurück. Wenn mehrere Clients den gleichen Inhalt abfragen, dann kann der Proxy diesen Inhalt aus seinem Zwischenspeicher ausliefern, anstatt ihn jedesmal vom ursprünglichen Server anzufordern, und dadurch die Antwortzeit verringern.

Siehe: [mod_proxy](#)

Public-Key-Kryptographie [ˈpʌblik ki: ˈkɪptɔɡrɑˈfi:]

Theorie und Anwendung asymmetrischer Verschlüsselungssysteme, die einen Schlüssel zur Verschlüsselung und einen anderen zur Entschlüsselung verwenden. Zwei derart zusammengehörende Schlüssel bilden Schlüssel-paar. Man spricht auch von "Asymmetrischer Kryptographie".

Siehe: [SSL/TLS-Verschlüsselung](#)

Regulärer Ausdruck (Regex)

Eine Form, ein Muster im Text zu beschreiben - zum Beispiel: "alle Wörter, die mit dem Buchstaben A beginnen" oder "Jeder Satz mit zwei Kommata und ohne großes Q". Beim Apache sind reguläre Ausdrücke hilfreich, da sie auf sehr flexible Art und Weise die Anwendung bestimmter Eigenschaften auf eine Auswahl von Dateien oder Ressourcen ermöglichen. - Zum Beispiel können alle .gif- und .jpg-Dateien eines Verzeichnis "images" mit `"/images/.*`

(jpg|gif)\$" beschrieben werden. Der Apache verwendet Perl-kompatible reguläre Ausdrücke, wie sie die [PCRE](#)-Bibliothek bereitstellt.

Reverse Proxy [ri:və:s 'prɔksi]

Ein [Proxy](#)-Server, der dem Client gegenüber als *ursprünglicher Server* erscheint. Dies ist nützlich, um den tatsächlichen Server aus Sicherheitsgründen oder zur Lastverteilung vor dem Client zu verstecken.

Secure Sockets Layer [si'kjuə 'sɔkits 'leɪə] (**SSL**)

Ein von der Firma Netscape Communications Corporation entwickeltes Protokoll zur allgemeinen Authentisierung und Verschlüsselung der Kommunikation über TCP/IP-Netzwerke. Die meistverbreitete Nutzung ist *HTTPS*, d.h. HyperText Transfer Protocol (HTTP) über SSL.

Siehe: [SSL/TLS-Verschlüsselung](#)

Server Side Includes [sə:ə saɪd in'klu:ds] (**SSI**)

Eine Technik zum Einbetten von weiterverarbeitenden Anweisungen in HTML-Dateien.

Siehe: [Einführung in Server Side Includes](#)

Session ['seʃən]

Allgemein der Kontext einer Kommunikation.

SSLeay

Die Bibliothek der Original-SSL/TLS-Implementation von Eric A. Young

Symmetrische Kryptographie

Die Theorie und Anwendung von *Chiffren*, die einen einzigen geheimen Schlüssel sowohl zur Verschlüsselung als auch zur Entschlüsselung benutzen.

Siehe: [SSL/TLS-Verschlüsselung](#)

Tarball [ta:bɔ:l]

Ein Paket von Dateien, die mit dem Hilfsprogramm tar

zusammengefasst wurden. Apache-Distributionen werden in komprimierten tar-Archiven oder unter Verwendung von `pkzip` gespeichert.

Transport Layer Security [træns'pɔ:t 'eiə si'kjʊəriti] **(TLS)**

Das SSL-Nachfolgeprotokoll, das von der Internet Engineering Task Force (IETF) zur allgemeinen Authentisierung und Verschlüsselung einer Kommunikation über TCP/IP-Netzwerke entwickelt worden ist. TLS Version 1 ist nahezu identisch mit SSL Version 3.

Siehe: [SSL/TLS-Verschlüsselung](#)

Umgebungsvariable (env-Variable)

Benannte, von der Betriebssystem-Shell verwaltete Variablen zur Speicherung von Informationen und zur Kommunikation zwischen Programmen. Der Apache beinhaltet auch interne Variablen, die ebenfalls Umgebungsvariablen genannt werden, die aber statt in der Shell-Umgebung in internen Apache-Strukturen gespeichert sind.

Siehe: [Umgebungsvariablen im Apache](#)

Uniform Resource Locator ['ju:nifo:m ri'sɔ:s læu'keitə] **(URL)**

Der Name bzw. die Adresse einer Ressource im Internet. Dies ist der allgemein gebräuchliche Ausdruck für die formale Bezeichnung [Uniform Resource Identifier](#). URLs bestehen üblicherweise aus einem Schema wie `http` oder `https`, einem Hostnamen und einem Pfad. Die URL für diese Seite könnte `http://httpd.apache.org/docs/2.2/glossary.html` lauten.

Uniform Resource Identifier ['ju:nifo:m ri'sɔ:s ai'dentifaiə] **(URI)**

Eine kompakte Zeichenfolge zur Identifizierung einer abstrakten oder physischen Ressource. Er wird in dem [RFC 2396](#) formell definiert. Im World Wide Web verwendete URIs werden üblicherweise als [URLs](#) bezeichnet.

Virtual-Hosting [və'tjuəl həustɪŋ]

Die Bedienung mehrere Websites mit einer einzigen Apache-Instanz. *IP-basierte virtuelle Hosts* unterscheiden zwischen

verschiedenen Websites aufgrund ihrer IP-Adressen, während *namensbasierte virtuelle Hosts* nur den Namen des Hosts verwenden und daher mehrere Angebote unter der gleichen IP-Adresse hosten können.

Siehe: [Apache-Dokumentation zu virtuellen Hosts](#)

Voll-qualifizierter Domainname (FQDN)

Der eindeutige Name einer Netzwerkeinheit, bestehend aus einem Hostnamen und dem Domainnamen, welcher zu einer IP-Adresse aufgelöst werden kann. Zum Beispiel ist `www` ein Hostname, `example.com` ein Domainname und `www.example.com` ein voll-qualifizierter Domainname.

Website [websait]

Im Gegensatz zur Webseite, die einer konkreten URL entspricht, ist mit Website ein komplettes Angebot unter einem bestimmten Hostnamen (und Port) gemeint. Dieses kann aus vielen verschiedenen Webseiten bestehen.

X.509

Ein von der International Telecommunication Union (ITU-T) empfohlenes Schema für Authentifizierungszertifikate. Es wird für SSL/TLS-Authentifizierungen verwendet.

Siehe: [SSL/TLS-Verschlüsselung](#)

Zertifikat

Ein Datensatz zur [Authentisierung](#) einer Netzwerkeinheit wie Server oder Client. Ein Zertifikat enthält [X.509](#)-Informationen über seinen Eigentümer (das sogenannte *Betreff* (*Anm.d.Ü.:* engl.: *subject*)) und die signierende [Certification Authority](#) (der sogenannte *Aussteller* (*Anm.d.Ü.:* engl.: *issuer*)) sowie den [öffentlichen Schlüssel](#) des Eigentümers und die Signatur der CA. Netzwerkeinheiten überprüfen diese Signatur mit Hilfe von CA-Zertifikaten.

Siehe: [SSL/TLS-Verschlüsselung](#)

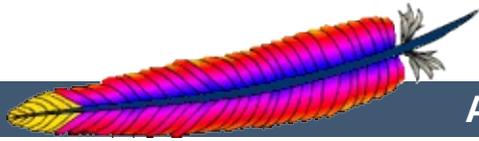
Zugriffskontrolle

Die Beschränkung des Zugriffs auf Netzwerkbereiche. Im Apache-

Kontext in der Regel die Zugriffsbeschränkung auf bestimmte *URLs*.
Siehe: [Authentisierung, Autorisierung und Zugriffskontrolle](#)

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Verzeichnis der Direktiven

Hier sind alle Apache-Direktiven aufgeführt, die in der Standard-Apache-Distribution verfügbar sind. Sie sind in einem einheitlichen Format beschrieben. Ein [Glossar](#) erläutert die in der Beschreibung verwendeten Begriffe.

Außerdem existiert eine [Kurzreferenz der Direktiven](#), welche zu jeder Direktive eine Zusammenfassung der Details enthält.

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) |
[R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#)

- [AcceptFilter](#)
- [AcceptMutex](#)
- [AcceptPathInfo](#)
- [AccessFileName](#)
- [Action](#)
- [AddAlt](#)
- [AddAltByEncoding](#)
- [AddAltByType](#)
- [AddCharset](#)
- [AddDefaultCharset](#)
- [AddDescription](#)
- [AddEncoding](#)
- [AddHandler](#)
- [AddIcon](#)
- [AddIconByEncoding](#)
- [AddIconByType](#)
- [AddInputFilter](#)
- [AddLanguage](#)
- [AddModuleInfo](#)
- [AddOutputFilter](#)
- [AddOutputFilterByType](#)

- [AddType](#)
- [Alias](#)
- [AliasMatch](#)
- [Allow](#)
- [AllowCONNECT](#)
- [AllowEncodedSlashes](#)
- [AllowOverride](#)
- [Anonymous](#)
- [Anonymous_LogEmail](#)
- [Anonymous_MustGiveEmail](#)
- [Anonymous_NoUserID](#)
- [Anonymous_VerifyEmail](#)
- [AuthBasicAuthoritative](#)
- [AuthBasicProvider](#)
- [AuthDBDUserPWQuery](#)
- [AuthDBDUserRealmQuery](#)
- [AuthDBMGroupFile](#)
- [AuthDBMType](#)
- [AuthDBMUserFile](#)
- [AuthDefaultAuthoritative](#)
- [AuthDigestAlgorithm](#)
- [AuthDigestDomain](#)
- [AuthDigestNcCheck](#)
- [AuthDigestNonceFormat](#)
- [AuthDigestNonceLifetime](#)
- [AuthDigestProvider](#)
- [AuthDigestQop](#)
- [AuthDigestShmemSize](#)
- [AuthGroupFile](#)
- [AuthLDAPBindAuthoritative](#)
- [AuthLDAPBindDN](#)
- [AuthLDAPBindPassword](#)
- [AuthLDAPCharsetConfig](#)
- [AuthLDAPCompareDNOnServer](#)

- [AuthLDAPDereferenceAliases](#)
- [AuthLDAPGroupAttribute](#)
- [AuthLDAPGroupAttributeIsDN](#)
- [AuthLDAPRemoteUserAttribute](#)
- [AuthLDAPRemoteUserIsDN](#)
- [AuthLDAPUrl](#)
- [AuthName](#)
- [<AuthnProviderAlias>](#)
- [AuthType](#)
- [AuthUserFile](#)
- [AuthzDBMAuthoritative](#)
- [AuthzDBMType](#)
- [AuthzDefaultAuthoritative](#)
- [AuthzGroupFileAuthoritative](#)
- [AuthzLDAPAuthoritative](#)
- [AuthzOwnerAuthoritative](#)
- [AuthzUserAuthoritative](#)
- [BalancerMember](#)
- [BrowserMatch](#)
- [BrowserMatchNoCase](#)
- [BufferedLogs](#)
- [CacheDefaultExpire](#)
- [CacheDirLength](#)
- [CacheDirLevels](#)
- [CacheDisable](#)
- [CacheEnable](#)
- [CacheFile](#)
- [CacheIgnoreCacheControl](#)
- [CacheIgnoreHeaders](#)
- [CacheIgnoreNoLastMod](#)
- [CacheIgnoreQueryString](#)
- [CacheIgnoreURLSessionIdentifiers](#)
- [CacheLastModifiedFactor](#)
- [CacheLock](#)

- [CacheLockMaxAge](#)
- [CacheLockPath](#)
- [CacheMaxExpire](#)
- [CacheMaxFileSize](#)
- [CacheMinFileSize](#)
- [CacheNegotiatedDocs](#)
- [CacheRoot](#)
- [CacheStoreNoStore](#)
- [CacheStorePrivate](#)
- [CGIDScriptTimeout](#)
- [CGIMapExtension](#)
- [CharsetDefault](#)
- [CharsetOptions](#)
- [CharsetSourceEnc](#)
- [CheckCaseOnly](#)
- [CheckSpelling](#)
- [ChrootDir](#)
- [ContentDigest](#)
- [CookieDomain](#)
- [CookieExpires](#)
- [CookieLog](#)
- [CookieName](#)
- [CookieStyle](#)
- [CookieTracking](#)
- [CoreDumpDirectory](#)
- [CustomLog](#)
- [Dav](#)
- [DavDepthInfinity](#)
- [DavGenericLockDB](#)
- [DavLockDB](#)
- [DavMinTimeout](#)
- [DBDExptime](#)
- [DBDKeep](#)
- [DBDMax](#)

- [DBDMin](#)
- [DBDParams](#)
- [DBDPersist](#)
- [DBDPrepareSQL](#)
- [DBDriver](#)
- [DefaultIcon](#)
- [DefaultLanguage](#)
- [DefaultType](#)
- [DeflateBufferSize](#)
- [DeflateCompressionLevel](#)
- [DeflateFilterNote](#)
- [DeflateInflateLimitRequestBody](#)
- [DeflateInflateRatioBurst](#)
- [DeflateInflateRatioLimit](#)
- [DeflateMemLevel](#)
- [DeflateWindowSize](#)
- [Deny](#)
- [<Directory>](#)
- [DirectoryIndex](#)
- [<DirectoryMatch>](#)
- [DirectorySlash](#)
- [DocumentRoot](#)
- [DumpIOInput](#)
- [DumpIOLogLevel](#)
- [DumpIOOutput](#)
- [EnableExceptionHook](#)
- [EnableMMAP](#)
- [EnableSendfile](#)
- [ErrorDocument](#)
- [ErrorLog](#)
- [Example](#)
- [ExpiresActive](#)
- [ExpiresByType](#)
- [ExpiresDefault](#)

- [ExtendedStatus](#)
- [ExtFilterDefine](#)
- [ExtFilterOptions](#)
- [FallbackResource](#)
- [FileETag](#)
- [<Files>](#)
- [<FilesMatch>](#)
- [FilterChain](#)
- [FilterDeclare](#)
- [FilterProtocol](#)
- [FilterProvider](#)
- [FilterTrace](#)
- [ForceLanguagePriority](#)
- [ForceType](#)
- [ForensicLog](#)
- [GprofDir](#)
- [GracefulShutdownTimeout](#)
- [Group](#)
- [Header](#)
- [HeaderName](#)
- [HostnameLookups](#)
- [HttpProtocolOptions](#)
- [IdentityCheck](#)
- [IdentityCheckTimeout](#)
- [<IfDefine>](#)
- [<IfModule>](#)
- [<IfVersion>](#)
- [ImapBase](#)
- [ImapDefault](#)
- [ImapMenu](#)
- [Include](#)
- [IndexHeadInsert](#)
- [IndexIgnore](#)
- [IndexOptions](#)

- [IndexOrderDefault](#)
- [IndexStyleSheet](#)
- [ISAPIAppendLogToErrors](#)
- [ISAPIAppendLogToQuery](#)
- [ISAPICacheFile](#)
- [ISAPIFakeAsync](#)
- [ISAPILogNotSupported](#)
- [ISAPIReadAheadBuffer](#)
- [KeepAlive](#)
- [KeepAliveTimeout](#)
- [LanguagePriority](#)
- [LDAPCacheEntries](#)
- [LDAPCacheTTL](#)
- [LDAPConnectionTimeout](#)
- [LDAPOpCacheEntries](#)
- [LDAPOpCacheTTL](#)
- [LDAPSharedCacheFile](#)
- [LDAPSharedCacheSize](#)
- [LDAPTrustedClientCert](#)
- [LDAPTrustedGlobalCert](#)
- [LDAPTrustedMode](#)
- [LDAPVerifyServerCert](#)
- [<Limit>](#)
- [<LimitExcept>](#)
- [LimitInternalRecursion](#)
- [LimitRequestBody](#)
- [LimitRequestFields](#)
- [LimitRequestFieldSize](#)
- [LimitRequestLine](#)
- [LimitXMLRequestBody](#)
- [Listen](#)
- [ListenBackLog](#)
- [LoadFile](#)
- [LoadModule](#)

- [<Location>](#)
- [<LocationMatch>](#)
- [LockFile](#)
- [LogFormat](#)
- [LogLevel](#)
- [MaxClients](#)
- [MaxKeepAliveRequests](#)
- [MaxMemFree](#)
- [MaxRanges](#)
- [MaxRequestsPerChild](#)
- [MaxRequestsPerThread](#)
- [MaxSpareServers](#)
- [MaxSpareThreads](#)
- [MaxThreads](#)
- [MCacheMaxObjectCount](#)
- [MCacheMaxObjectSize](#)
- [MCacheMaxStreamingBuffer](#)
- [MCacheMinObjectSize](#)
- [MCacheRemovalAlgorithm](#)
- [MCacheSize](#)
- [MergeTrailers](#)
- [MetaDir](#)
- [MetaFiles](#)
- [MetaSuffix](#)
- [MimeMagicFile](#)
- [MinSpareServers](#)
- [MinSpareThreads](#)
- [MMapFile](#)
- [ModMimeUsePathInfo](#)
- [MultiviewsMatch](#)
- [NameVirtualHost](#)
- [NoProxy](#)
- [NWSSLTrustedCerts](#)
- [NWSSLUpgradeable](#)

- [Options](#)
- [Order](#)
- [PassEnv](#)
- [PidFile](#)
- [Protocol](#)
- [ProtocolEcho](#)
- [<Proxy>](#)
- [ProxyBadHeader](#)
- [ProxyBlock](#)
- [ProxyDomain](#)
- [ProxyErrorOverride](#)
- [ProxyFtpDirCharset](#)
- [ProxyIOBufferSize](#)
- [<ProxyMatch>](#)
- [ProxyMaxForwards](#)
- [ProxyPass](#)
- [ProxyPassInterpolateEnv](#)
- [ProxyPassMatch](#)
- [ProxyPassReverse](#)
- [ProxyPassReverseCookieDomain](#)
- [ProxyPassReverseCookiePath](#)
- [ProxyPreserveHost](#)
- [ProxyReceiveBufferSize](#)
- [ProxyRemote](#)
- [ProxyRemoteMatch](#)
- [ProxyRequests](#)
- [ProxySCGIInternalRedirect](#)
- [ProxySCGISendfile](#)
- [ProxySet](#)
- [ProxyStatus](#)
- [ProxyTimeout](#)
- [ProxyVia](#)
- [ReadmeName](#)
- [ReceiveBufferSize](#)

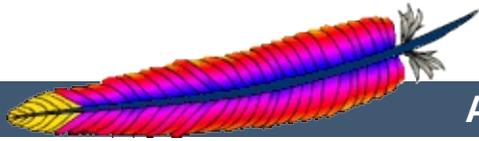
- [Redirect](#)
- [RedirectMatch](#)
- [RedirectPermanent](#)
- [RedirectTemp](#)
- [RegisterHttpMethod](#)
- [RemoveCharset](#)
- [RemoveEncoding](#)
- [RemoveHandler](#)
- [RemoveInputFilter](#)
- [RemoveLanguage](#)
- [RemoveOutputFilter](#)
- [RemoveType](#)
- [RequestHeader](#)
- [RequestReadTimeout](#)
- [Require](#)
- [RewriteBase](#)
- [RewriteCond](#)
- [RewriteEngine](#)
- [RewriteLock](#)
- [RewriteLog](#)
- [RewriteLogLevel](#)
- [RewriteMap](#)
- [RewriteOptions](#)
- [RewriteRule](#)
- [RLimitCPU](#)
- [RLimitMEM](#)
- [RLimitNPROC](#)
- [Satisfy](#)
- [ScoreBoardFile](#)
- [Script](#)
- [ScriptAlias](#)
- [ScriptAliasMatch](#)
- [ScriptInterpreterSource](#)
- [ScriptLog](#)

- [ScriptLogBuffer](#)
- [ScriptLogLength](#)
- [ScriptSock](#)
- [SecureListen](#)
- [SeeRequestTail](#)
- [SendBufferSize](#)
- [ServerAdmin](#)
- [ServerAlias](#)
- [ServerLimit](#)
- [ServerName](#)
- [ServerPath](#)
- [ServerRoot](#)
- [ServerSignature](#)
- [ServerTokens](#)
- [SetEnv](#)
- [SetEnvIf](#)
- [SetEnvIfNoCase](#)
- [SetHandler](#)
- [SetInputFilter](#)
- [SetOutputFilter](#)
- [SSIEnableAccess](#)
- [SSIEndTag](#)
- [SSIErrorMsg](#)
- [SSIETag](#)
- [SSILastModified](#)
- [SSIStartTag](#)
- [SSITimeFormat](#)
- [SSIUndefinedEcho](#)
- [SSLCACertificateFile](#)
- [SSLCACertificatePath](#)
- [SSLCADNRequestFile](#)
- [SSLCADNRequestPath](#)
- [SSLCARevocationFile](#)
- [SSLCARevocationPath](#)

- [SSLCertificateChainFile](#)
- [SSLCertificateFile](#)
- [SSLCertificateKeyFile](#)
- [SSLCipherSuite](#)
- [SSLCompression](#)
- [SSLCryptoDevice](#)
- [SSLEngine](#)
- [SSLFIPS](#)
- [SSLHonorCipherOrder](#)
- [SSLInsecureRenegotiation](#)
- [SSLMutex](#)
- [SSLOptions](#)
- [SSLPassPhraseDialog](#)
- [SSLProtocol](#)
- [SSLProxyCACertificateFile](#)
- [SSLProxyCACertificatePath](#)
- [SSLProxyCARevocationFile](#)
- [SSLProxyCARevocationPath](#)
- [SSLProxyCheckPeerCN](#)
- [SSLProxyCheckPeerExpire](#)
- [SSLProxyCipherSuite](#)
- [SSLProxyEngine](#)
- [SSLProxyMachineCertificateChainFile](#)
- [SSLProxyMachineCertificateFile](#)
- [SSLProxyMachineCertificatePath](#)
- [SSLProxyProtocol](#)
- [SSLProxyVerify](#)
- [SSLProxyVerifyDepth](#)
- [SSLRandomSeed](#)
- [SSLRenegBufferSize](#)
- [SSLRequire](#)
- [SSLRequireSSL](#)
- [SSLSessionCache](#)
- [SSLSessionCacheTimeout](#)

- [SSLSessionTicketKeyFile](#)
- [SSLSessionTickets](#)
- [SSLStrictSNIVHostCheck](#)
- [SSLUserName](#)
- [SSLVerifyClient](#)
- [SSLVerifyDepth](#)
- [StartServers](#)
- [StartThreads](#)
- [Substitute](#)
- [SubstituteInheritBefore](#)
- [Suexec](#)
- [SuexecUserGroup](#)
- [ThreadLimit](#)
- [ThreadsPerChild](#)
- [ThreadStackSize](#)
- [TimeOut](#)
- [TraceEnable](#)
- [TransferLog](#)
- [TypesConfig](#)
- [UnsetEnv](#)
- [UseCanonicalName](#)
- [UseCanonicalPhysicalPort](#)
- [User](#)
- [UserDir](#)
- [VirtualDocumentRoot](#)
- [VirtualDocumentRootIP](#)
- [<VirtualHost>](#)
- [VirtualScriptAlias](#)
- [VirtualScriptAliasIP](#)
- [Win32DisableAcceptEx](#)
- [XBitHack](#)

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#) > [Module](#)

Kurzreferenz der Direktiven

Die Kurzreferenz der Direktiven zeigt die Verwendung, Voreinstellung, den Status und den Kontext aller Apache-Konfigurationsanweisungen. Für weitergehende Informationen schauen Sie bitte im [Verzeichnis der Direktiven](#).

Die erste Spalte enthält den Namen und die Verwendung. Die zweite Spalte zeigt die Voreinstellung der Direktive, sofern eine Voreinstellung existiert. Wenn die Voreinstellung zu breit für die Anzeige ist, wird sie abgeschnitten und mit einem nachfolgenden "+" versehen.

Die dritte und vierte Spalte geben den Kontext an, in dem die Direktive erlaubt ist, sowie den Status der Direktive entsprechend der Legende.

A	B	C	D	E	F	s Serverkonfiguration	C Core
G	H	I	K	L	M	v Virtual Host	M MPM
N	O	P	R	S	T	d Verzeichnis	B Basis
U	V	W	X			h .htaccess	E Erweiterung
							X experimentell

AcceptFilter <i>protocol accept filter</i>	
Configures optimizations for a Protocol's Listener Sockets	
AcceptMutex <i>Default Methode</i>	Default
Vom Apache verwendete Methode zur Serialisierung mehrerer Kindprozesse, die Anfragen Sockets entgegennehmen.	
AcceptPathInfo <i>On Off Default</i>	Default
Ressourcen lassen angehängte Pfadangaben zu	
AccessFileName <i>Dateiname [Dateiname] ...</i>	.htaccess
Name der dezentralen Konfigurationsdateien	
Action <i>Aktionsart CGI-Skript [virtual]</i>	
Aktiviert ein CGI-Skript für einen bestimmten Handler oder Content-Type	

AddAlt <i>string file [file] ...</i>	
Alternate text to display for a file, instead of an icon selected by filename	
AddAltByEncoding <i>string MIME-encoding [MIME-encoding] ...</i>	
Alternate text to display for a file instead of an icon selected by MIME-encoding	
AddAltByType <i>string MIME-type [MIME-type] ...</i>	
Alternate text to display for a file, instead of an icon selected by MIME content-type	
AddCharset <i>charset extension [extension] ...</i>	
Maps the given filename extensions to the specified content charset	
AddDefaultCharset On Off <i>Zeichenkodierung</i>	Off
Standard-Charset-Parameter, der bei Antworten vom Content-Type text/plain oder tex hinzugefügt wird	
AddDescription <i>string file [file] ...</i>	
Description to display for a file	
AddEncoding <i>MIME-enc extension [extension] ...</i>	
Maps the given filename extensions to the specified encoding type	
AddHandler <i>handler-name extension [extension] ...</i>	
Maps the filename extensions to the specified handler	
AddIcon <i>icon name [name] ...</i>	
Icon to display for a file selected by name	
AddIconByEncoding <i>icon MIME-encoding [MIME-encoding] ...</i>	
Icon to display next to files selected by MIME content-encoding	
AddIconByType <i>icon MIME-type [MIME-type] ...</i>	
Icon to display next to files selected by MIME content-type	
AddInputFilter <i>filter[:filter...] extension [extension] ...</i>	
Maps filename extensions to the filters that will process client requests	
AddLanguage <i>MIME-lang extension [extension] ...</i>	
Maps the given filename extension to the specified content language	

AddModuleInfo <i>module-name string</i>	
Adds additional information to the module information displayed by the server-info handler	
AddOutputFilter <i>filter[:filter...] extension [extension] ...</i>	
Maps filename extensions to the filters that will process responses from the server	
AddOutputFilterByType <i>Filter[:Filter...] MIME-Type [MIME-Type] ...</i>	
einen Ausgabefilter einem bestimmten MIME-Type zuordnen	
AddType <i>MIME-type extension [extension] ...</i>	
Maps the given filename extensions onto the specified content type	
Alias <i>URL-path file-path directory-path</i>	
Maps URLs to filesystem locations	
AliasMatch <i>regex file-path directory-path</i>	
Maps URLs to filesystem locations using regular expressions	
Allow from <i>all host env=[!]env-variable [host env=[!]env-variable] ...</i>	
Controls which hosts can access an area of the server	
AllowCONNECT <i>port [port] ...</i>	443 563
Ports that are allowed to CONNECT through the proxy	
AllowEncodedSlashes <i>On Off</i>	Off
Legt fest, ob kodierte Pfadtrennzeichen in URLs durchgereicht werden dürfen	
AllowOverride <i>All None Direktiven-Typ [Direktiven-Typ] ...</i>	All
Direktiven-Typen, die in .htaccess-Dateien erlaubt sind.	
Anonymous <i>user [user] ...</i>	
Specifies userIDs that are allowed access without password verification	
Anonymous_ <i>LogEmail On Off</i>	On
Sets whether the password entered will be logged in the error log	
Anonymous_ <i>MustGiveEmail On Off</i>	On
Specifies whether blank passwords are allowed	
Anonymous_ <i>NoUserID On Off</i>	Off
Sets whether the userID field may be empty	
Anonymous_ <i>VerifyEmail On Off</i>	Off
Sets whether to check the password field for a correctly formatted email address	
AuthBasicAuthoritative <i>On Off</i>	On

	Sets whether authorization and authentication are passed to lower level modules	
AuthBasicProvider <i>provider-name</i> [<i>provider-name</i>] ...		file
	Sets the authentication provider(s) for this location	
AuthDBDUserPWQuery <i>query</i>		
	SQL query to look up a password for a user	
AuthDBDUserRealmQuery <i>query</i>		
	SQL query to look up a password hash for a user and realm.	
AuthDBMGroupFile <i>file-path</i>		
	Sets the name of the database file containing the list of user groups for authorization	
AuthDBMType default SDBM GDBM NDBM DB		default
	Sets the type of database file that is used to store passwords	
AuthDBMUserFile <i>file-path</i>		
	Sets the name of a database file containing the list of users and passwords for authentication	
AuthDefaultAuthoritative On Off		On
	Sets whether authentication is passed to lower level modules	
AuthDigestAlgorithm MD5 MD5-sess		MD5
	Selects the algorithm used to calculate the challenge and response hashes in digest authentication	
AuthDigestDomain <i>URI</i> [<i>URI</i>] ...		
	URIs that are in the same protection space for digest authentication	
AuthDigestNcCheck On Off		Off
	Enables or disables checking of the nonce-count sent by the server	
AuthDigestNonceFormat <i>format</i>		
	Determines how the nonce is generated	
AuthDigestNonceLifetime <i>seconds</i>		300
	How long the server nonce is valid	
AuthDigestProvider <i>provider-name</i> [<i>provider-name</i>] ...		file
	Sets the authentication provider(s) for this location	
AuthDigestQop none auth auth-int [auth auth-int]		auth
	Determines the quality-of-protection to use in digest authentication	
AuthDigestShmemSize <i>size</i>		1000
	The amount of shared memory to allocate for keeping track of clients	

<u>AuthGroupFile</u> <i>file-path</i>	
Sets the name of a text file containing the list of user groups for authorization	
<u>AuthLDAPBindAuthoritative</u> <i>off on</i>	on
Determines if other authentication providers are used when a user can be mapped to a DN cannot successfully bind with the user's credentials.	
<u>AuthLDAPBindDN</u> <i>distinguished-name</i>	
Optional DN to use in binding to the LDAP server	
<u>AuthLDAPBindPassword</u> <i>password</i>	
Password used in conjunction with the bind DN	
<u>AuthLDAPCharsetConfig</u> <i>file-path</i>	
Language to charset conversion configuration file	
<u>AuthLDAPCompareDNOnServer</u> <i>on off</i>	on
Use the LDAP server to compare the DNS	
<u>AuthLDAPDereferenceAliases</u> <i>never searching finding always</i>	Always
When will the module de-reference aliases	
<u>AuthLDAPGroupAttribute</u> <i>attribute</i>	member uniquememb +
LDAP attributes used to check for group membership	
<u>AuthLDAPGroupAttributeIsDN</u> <i>on off</i>	on
Use the DN of the client username when checking for group membership	
<u>AuthLDAPRemoteUserAttribute</u> <i>uid</i>	
Use the value of the attribute returned during the user query to set the REMOTE_USER environment variable	
<u>AuthLDAPRemoteUserIsDN</u> <i>on off</i>	off
Use the DN of the client username to set the REMOTE_USER environment variable	
<u>AuthLDAPUrl</u> <i>url [NONE SSL TLS STARTTLS]</i>	
URL specifying the LDAP search parameters	
<u>AuthName</u> <i>auth-Bereich</i>	
Autorisierungsbereich zur Verwendung in der HTTP-Authentisierung	
<u><AuthnProviderAlias</u> <i>baseProvider Alias</i> <u>> ...</u> <u></AuthnProviderAlias</u> <u>></u>	
Enclose a group of directives that represent an extension of a base authentication provider by the specified alias	
<u>AuthType</u> <i>Basic Digest</i>	
Art der Authentisierung	

<u>AuthUserFile</u> <i>file-path</i>	
Sets the name of a text file containing the list of users and passwords for authentication	
<u>AuthzDBMAuthoritative</u> On Off	On
Sets whether authorization will be passed on to lower level modules	
<u>AuthzDBMType</u>	default
<u>default SDBM GDBM NDBM DB</u>	
Sets the type of database file that is used to store list of user groups	
<u>AuthzDefaultAuthoritative</u> On Off	On
Sets whether authorization is passed to lower level modules	
<u>AuthzGroupFileAuthoritative</u> On Off	On
Sets whether authorization will be passed on to lower level modules	
<u>AuthzLDAPAuthoritative</u> on off	on
Prevent other authentication modules from authenticating the user if this one fails	
<u>AuthzOwnerAuthoritative</u> On Off	On
Sets whether authorization will be passed on to lower level modules	
<u>AuthzUserAuthoritative</u> On Off	On
Sets whether authorization will be passed on to lower level modules	
<u>BalancerMember</u> [<i>balancerurl</i>] <i>url</i> [<i>key=value</i> [<i>key=value ...</i>]]	
Add a member to a load balancing group	
<u>BrowserMatch</u> <i>regex</i> [!] <i>env-variable</i> [= <i>value</i>] [!] <i>env-variable</i> [= <i>value</i>] ...	
Sets environment variables conditional on HTTP User-Agent	
<u>BrowserMatchNoCase</u> <i>regex</i> [!] <i>env-variable</i> [= <i>value</i>] [!] <i>env-variable</i> [= <i>value</i>] ...	
Sets environment variables conditional on User-Agent without respect to case	
<u>BufferedLogs</u> On Off	Off
Buffer log entries in memory before writing to disk	
<u>CacheDefaultExpire</u> <i>seconds</i>	3600 (one hour)
The default duration to cache a document when no expiry date is specified.	
<u>CacheDirLength</u> <i>length</i>	2
The number of characters in subdirectory names	
<u>CacheDirLevels</u> <i>levels</i>	3
The number of levels of subdirectories in the cache.	
<u>CacheDisable</u> <i>url-string</i>	

CacheEnable <i>cache_type url-string</i>	
Disable caching of specified URLs	
CacheFile <i>file-path [file-path] ...</i>	
Enable caching of specified URLs using a specified storage manager	
CacheIgnoreCacheControl On Off	Off
Cache a list of file handles at startup time	
CacheIgnoreHeaders <i>header-string [header-string] ...</i>	None
Ignore request to not serve cached content to client	
CacheIgnoreNoLastMod On Off	Off
Do not store the given HTTP header(s) in the cache.	
CacheIgnoreQueryString On Off	Off
Ignore the fact that a response has no Last Modified header.	
CacheIgnoreURLSessionIdentifiers <i>identifier [identifier] ...</i>	None
Ignore query string when caching	
CacheLastModifiedFactor <i>float</i>	0.1
Ignore defined session identifiers encoded in the URL when caching	
CacheLock <i>on off</i>	off
The factor used to compute an expiry date based on the LastModified date.	
CacheLockMaxAge <i>integer</i>	5
Enable the thundering herd lock.	
CacheLockPath <i>directory</i>	/tmp/mod_cache-lock
Set the maximum possible age of a cache lock.	
CacheMaxExpire <i>seconds</i>	86400 (one day)
Set the lock path directory.	
CacheMaxFileSize <i>bytes</i>	1000000
The maximum time in seconds to cache a document	
CacheMinFileSize <i>bytes</i>	1
The maximum size (in bytes) of a document to be placed in the cache	
CacheNegotiatedDocs On Off	Off
The minimum size (in bytes) of a document to be placed in the cache	
CacheRoot <i>directory</i>	
Allows content-negotiated documents to be cached by proxy servers	
The directory root under which cache files are stored	

CacheStoreNoStore On Off	Off
Attempt to cache requests or responses that have been marked as no-store.	
CacheStorePrivate On Off	Off
Attempt to cache responses that the server has marked as private	
CGIDScriptTimeout <i>time[s ms]</i>	
The length of time to wait for more output from the CGI program	
CGIMapExtension <i>CGI-Pfad .Endung</i>	
Technik zur Bestimmung des Interpreters für CGI-Skripte	
CharsetDefault <i>charset</i>	
Charset to translate into	
CharsetOptions <i>option [option] ...</i>	DebugLevel=0 NoImp
Configures charset translation behavior	
CharsetSourceEnc <i>charset</i>	
Source charset of files	
CheckCaseOnly on off	Off
Limits the action of the spelling module to case corrections	
CheckSpelling on off	Off
Enables the spelling module	
ChrootDir <i>/path/to/directory</i>	
Directory for apache to run chroot(8) after startup.	
ContentDigest On Off	Off
Aktiviert die Generierung von Content - MD5 HTTP-Response-Headern	
CookieDomain <i>domain</i>	
The domain to which the tracking cookie applies	
CookieExpires <i>expiry-period</i>	
Expiry time for the tracking cookie	
CookieLog <i>filename</i>	
Sets filename for the logging of cookies	
CookieName <i>token</i>	Apache
Name of the tracking cookie	
CookieStyle	Netscape
Netscape Cookie Cookie2 RFC2109 RFC2965	
Format of the cookie header field	
CookieTracking on off	off
Enables tracking cookie	

CoreDumpDirectory <i>Verzeichnis</i>	
Verzeichnis, in das der Apache zu wechseln versucht, bevor er einen Hauptspeicherauszug	
CustomLog <i>file pipe format nickname</i> [<i>env=</i> <i>![environment-variable]</i>]	
Sets filename and format of log file	
Dav <i>On Off provider-name</i>	Off
Enable WebDAV HTTP methods	
DavDepthInfinity <i>on off</i>	off
Allow PROPFIND, Depth: Infinity requests	
DavGenericLockDB <i>file-path</i>	
Location of the DAV lock database	
DavLockDB <i>file-path</i>	
Location of the DAV lock database	
DavMinTimeout <i>seconds</i>	0
Minimum amount of time the server holds a lock on a DAV resource	
DBDExptime <i>time-in-seconds</i>	300
Keepalive time for idle connections	
DBDKeep <i>number</i>	2
Maximum sustained number of connections	
DBDMax <i>number</i>	10
Maximum number of connections	
DBDMin <i>number</i>	1
Minimum number of connections	
DBDParams <i>param1=value1[,param2=value2]</i>	
Parameters for database connection	
DBDPersist <i>On Off</i>	
Whether to use persistent connections	
DBDPrepareSQL <i>"SQL statement" label</i>	
Define an SQL prepared statement	
DBDriver <i>name</i>	
Specify an SQL driver	
DefaultIcon <i>url-path</i>	
Icon to display for files when no specific icon is configured	
DefaultLanguage <i>MIME-lang</i>	
Sets all files in the given scope to the specified language	

DefaultType <i>MIME-Type</i>	text/plain
MIME-Content-Type, der gesendet wird, wenn der Server den Typ nicht auf andere Weise e	
DeflateBufferSize <i>value</i>	8096
Fragment size to be compressed at one time by zlib	
DeflateCompressionLevel <i>value</i>	
How much compression do we apply to the output	
DeflateFilterNote [<i>type</i>] <i>notename</i>	
Places the compression ratio in a note for logging	
DeflateInflateLimitRequestBody<i>value</i>	
Maximum size of inflated request bodies	
DeflateInflateRatioBurst <i>value</i>	
Maximum number of times the inflation ratio for request bodies can be crossed	
DeflateInflateRatioLimit <i>value</i>	
Maximum inflation ratio for request bodies	
DeflateMemLevel <i>value</i>	9
How much memory should be used by zlib for compression	
DeflateWindowSize <i>value</i>	15
Zlib compression window size	
Deny from all <i>host</i> <i>env</i>=[!]<i>env-variable</i> [<i>host</i> <i>env</i>=[!]<i>env-variable</i>] ...	
Controls which hosts are denied access to the server	
<Directory <i>Verzeichnispfad</i>> ... </Directory>	
Umschließt eine Gruppe von Direktiven, die nur auf das genannte Verzeichnis des Dateisys Unterverzeichnisse angewendet werden	
DirectoryIndex <i>local-url</i> [<i>local-url</i>] ...	index.html
List of resources to look for when the client requests a directory	
<DirectoryMatch <i>regex</i>> ... </DirectoryMatch>	
Umschließt eine Gruppe von Direktiven, die auf Verzeichnisse des Dateisystems und ihre Unterverzeichnisse abgebildet werden, welche auf einen regulären Ausdruck passen	
DirectorySlash On Off	On
Toggle trailing slash redirects on or off	
DocumentRoot <i>Verzeichnis</i>	/usr/local/apache/h +
Verzeichnis, welches den Haupt-Dokumentenbaum bildet, der im Web sichtbar ist.	
DumpIOInput On Off	Off
Dump all input data to the error log	

DumpIOLogLevel <i>level</i>	debug
Controls the logging level of the DumpIO output	
DumpIOOutput On Off	Off
Dump all output data to the error log	
EnableExceptionHandler On Off	Off
Aktiviert einen Hook, der nach einem Absturz noch Ausnahmefehler behandeln lassen kann	
EnableMMAP On Off	On
Verwende Memory-Mapping, um Dateien während der Auslieferung zu lesen	
EnableSendfile On Off	On
Verwende die sendfile-Unterstützung des Kernels, um Dateien an den Client auszuliefern	
ErrorDocument <i>Fehlercode Dokument</i>	
Das, was der Server im Fehlerfall an den Client zurückgibt	
ErrorLog <i>Dateiname syslog[:facility]</i>	logs/error_log (Uni +
Ablageort, an dem der Server Fehler protokolliert	
Example	
Demonstration directive to illustrate the Apache module API	
ExpiresActive On Off	
Enables generation of Expires headers	
ExpiresByType <i>MIME-type <code>seconds</i>	
Value of the Expires header configured by MIME type	
ExpiresDefault <i><code>seconds</i>	
Default algorithm for calculating expiration time	
ExtendedStatus On Off	Off
Keep track of extended status information for each request	
ExtFilterDefine <i>filtername parameters</i>	
Define an external filter	
ExtFilterOptions <i>option [option] ...</i>	DebugLevel=0 NoLog +
Configure <code>mod_ext_filter</code> options	
FallbackResource <i>disabled local-uri</i>	
Define a default URL for requests that don't map to a file	
FileETag <i>Komponente ...</i>	INode MTime Size
Dateiattribute, die zur Erstellung des HTTP-Response-Headers ETag verwendet werden	
<Files <i>Dateiname</i>> ... </Files>	
Enthält Direktiven, die sich nur auf passende Dateinamen beziehen	

<FilesMatch regex> ... </FilesMatch>	
Enthält Direktiven, die für Dateinamen gelten, die auf einen regulären Ausdruck passen	
FilterChain [+=-@!]filter-name ...	
Configure the filter chain	
FilterDeclare filter-name [type]	
Declare a smart filter	
FilterProtocol filter-name [provider-name]	
proto-flags	
Deal with correct HTTP protocol handling	
FilterProvider filter-name provider-name	
[req resp env]=dispatch match	
Register a content filter	
FilterTrace filter-name level	
Get debug/diagnostic information from <code>mod filter</code>	
ForceLanguagePriority None Prefer Fallback	Prefer
[Prefer Fallback]	
Action to take if a single acceptable document is not found	
ForceType MIME-Type None	
Erzwingt die Auslieferung aller passendenden Dateien mit dem angegebenen MIME-Content	
ForensicLog filename pipe	
Sets filename of the forensic log	
GprofDir /tmp/gprof/ /tmp/gprof/%	
Directory to write gmon.out profiling data to.	
GracefulShutdownTimeout seconds	
Specify a timeout after which a gracefully shutdown server will exit.	
Group Unix-Gruppe	#-1
Benutzergruppe, unter welcher der Server Anfragen beantwortet	
Header [condition]	
set append merge add unset echo edit header	
[value] [replacement] [early env=!]variable	
Configure HTTP response headers	
HeaderName filename	
Name of the file that will be inserted at the top of the index listing	
HostnameLookups On Off Double	Off
Aktiviert DNS-Lookups auf Client-IP-Adressen	

HttpProtocolOptions [Strict Unsafe] [RegisteredMethods LenientMethods] [Allow0.9 Require1.0]	Strict LenientMetho +
Modify restrictions on HTTP Request Messages	
IdentityCheck On Off	Off
Enables logging of the RFC 1413 identity of the remote user	
IdentityCheckTimeout <i>seconds</i>	30
Determines the timeout duration for ident requests	
<IfDefine [!]<i>Parametername</i>> ... </IfDefine>	
Schließt Direktiven ein, die nur ausgeführt werden, wenn eine Testbedingung beim Start wa	
<IfModule [!]<i>Modulname Modulbezeichner</i>> ... </IfModule>	
Schließt Direktiven ein, die abhängig vom Vorhandensein oder Fehlen eines speziellen Moc werden	
<IfVersion [[!]<i>operator</i>] <i>version</i>> ... </IfVersion>	
contains version dependent configuration	
ImapBase <i>map referer URL</i>	http://servername/
Default base for imagemap files	
ImapDefault <i>error nocontent map referer URL</i>	nocontent
Default action when an imagemap is called with coordinates that are not explicitly mapped	
ImapMenu <i>none formatted semiformatted unformatted</i>	
Action if no coordinates are given when calling an imagemap	
Include <i>Dateiname Verzeichnis</i>	
Fügt andere Konfigurationsdateien innerhalb der Server-Konfigurationsdatei ein	
IndexHeadInsert "<i>markup ...</i>"	
Inserts text in the HEAD section of an index page.	
IndexIgnore <i>file [file] ...</i>	". "
Adds to the list of files to hide when listing a directory	
IndexOptions [+ -]<i>option</i> [[+ -]<i>option</i>] ...	
Various configuration settings for directory indexing	
IndexOrderDefault <i>Ascending Descending Name Date Size Description</i>	Ascending Name
Sets the default ordering of the directory index	

IndexStyleSheet <i>url-path</i>	
Adds a CSS stylesheet to the directory index	
ISAPIAppendLogToErrors <i>on off</i>	off
Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the error log	
ISAPIAppendLogToQuery <i>on off</i>	on
Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field	
ISAPICacheFile <i>file-path [file-path] ...</i>	
ISAPI .dll files to be loaded at startup	
ISAPIFakeAsync <i>on off</i>	off
Fake asynchronous support for ISAPI callbacks	
ISAPILogNotSupported <i>on off</i>	off
Log unsupported feature requests from ISAPI extensions	
ISAPIReadAheadBuffer <i>size</i>	49152
Size of the Read Ahead Buffer sent to ISAPI extensions	
KeepAlive <i>On Off</i>	On
Aktiviert persistente HTTP-Verbindungen	
KeepAliveTimeout <i>Sekunden</i>	5
Zeitspanne, die der Server während persistenter Verbindungen auf nachfolgende Anfragen	
LanguagePriority <i>MIME-lang [MIME-lang] ...</i>	
The precedence of language variants for cases where the client does not express a prefer	
LDAPCacheEntries <i>number</i>	1024
Maximum number of entries in the primary LDAP cache	
LDAPCacheTTL <i>seconds</i>	600
Time that cached items remain valid	
LDAPConnectionTimeout <i>seconds</i>	
Specifies the socket connection timeout in seconds	
LDAPOpCacheEntries <i>number</i>	1024
Number of entries used to cache LDAP compare operations	
LDAPOpCacheTTL <i>seconds</i>	600
Time that entries in the operation cache remain valid	
LDAPSharedCacheFile <i>directory-path/filename</i>	
Sets the shared memory cache file	
LDAPSharedCacheSize <i>bytes</i>	500000
Size in bytes of the shared-memory cache	

<u>LDAPTrustedClientCert</u> <i>type directory-path/filename/nickname [password]</i>	
Sets the file containing or nickname referring to a per connection client certificate. Not all LDAP support per connection client certificates.	
<u>LDAPTrustedGlobalCert</u> <i>type directory-path/filename [password]</i>	
Sets the file or database containing global trusted Certificate Authority or global client certificate	
<u>LDAPTrustedMode</u> <i>type</i>	
Specifies the SSL/TLS mode to be used when connecting to an LDAP server.	
<u>LDAPVerifyServerCert</u> <i>On Off</i>	On
Force server certificate verification	
<u><Limit Methode [Methode] ... > ... </Limit></u>	
Beschränkt die eingeschlossenen Zugriffskontrollen auf bestimmte HTTP-Methoden	
<u><LimitExcept Methode [Methode] ... > ... </LimitExcept></u>	
Beschränkt Zugriffskontrollen auf alle HTTP-Methoden außer den genannten	
<u>LimitInternalRecursion</u> <i>Zahl [Zahl]</i>	10
Bestimmt die maximale Anzahl interner Umleitungen und verschachtelter Unteranfragen	
<u>LimitRequestBody</u> <i>Bytes</i>	0
Begrenzt die Gesamtgröße des vom Client gesendeten HTTP-Request-Body	
<u>LimitRequestFields</u> <i>Anzahl</i>	100
Begrenzt die Anzahl der HTTP-Request-Header, die vom Client entgegengenommen werden	
<u>LimitRequestFieldSize</u> <i>Bytes</i>	
Begrenzt die Länge des vom Client gesendeten HTTP-Request-Headers	
<u>LimitRequestLine</u> <i>Bytes</i>	8190
Begrenzt die Länge der vom Client entgegengenommenen HTTP-Anfragezeile	
<u>LimitXMLRequestBody</u> <i>Bytes</i>	1000000
Begrenzt die Größe eines XML-basierten Request-Bodys	
<u>Listen</u> <i>[IP-Adresse:]Port</i>	
IP-Adressen und Ports, an denen der Server lauscht	
<u>ListenBacklog</u> <i>backlog</i>	
Maximale Länge der Warteschlange schwebender Verbindungen	
<u>LoadFile</u> <i>filename [filename] ...</i>	
Link in the named object file or library	
<u>LoadModule</u> <i>module filename</i>	

	Links in the object file or library, and adds to the list of active modules
<Location URL-Pfad URL> ... </Location>	
	Wendet die enthaltenen Direktiven nur auf die entsprechenden URLs an
<LocationMatch regex> ... </LocationMatch>	
	Wendet die enthaltenen Direktiven nur auf URLs an, die auf reguläre Ausdrücke passen
LockFile Dateiname	logs/accept.lock
	Ablageort der Lock-Datei für die Serialisierung von entgegengenommenen Anfragen
LogFormat format nickname [nickname]	"%h %l %u %t \"%r\" -
	Describes a format for use in a log file
LogLevel Level	warn
	Steuert die Ausführlichkeit des Fehlerprotokolls
MaxClients Anzahl	
	Maximale Anzahl der Kindprozesse, die zur Bedienung von Anfragen gestartet wird
MaxKeepAliveRequests Anzahl	100
	Anzahl der Anfragen, die bei einer persistenten Verbindung zulässig sind
MaxMemFree KBytes	0
	Maximale Menge des Arbeitsspeichers, den die Haupt-Zuteilungsroutine verwalten darf, ohne aufzurufen
MaxRanges default unlimited none number-of-ranges	200
	Number of ranges allowed before returning the complete resource
MaxRequestsPerChild number	10000
	Obergrenze für die Anzahl von Anfragen, die ein einzelner Kindprozess während seines Lebens
MaxRequestsPerThread Anzahl	0
	Die maximale Anzahl von Anfragen, die ein einzelner Thread während seiner Lebensdauer
MaxSpareServers Anzahl	10
	Maximale Anzahl der unbeschäftigten Kindprozesse des Servers
MaxSpareThreads Anzahl	
	Maximale Anzahl unbeschäftigter Threads
MaxThreads number	2048
	Set the maximum number of worker threads
MCacheMaxObjectCount value	1009
	The maximum number of objects allowed to be placed in the cache
MCacheMaxObjectSize bytes	10000
	The maximum size (in bytes) of a document allowed in the cache

MCacheMaxStreamingBuffer <i>size in bytes</i>	the smaller of 1000 +
Maximum amount of a streamed response to buffer in memory before declaring the response	
MCacheMinObjectSize <i>bytes</i>	1
The minimum size (in bytes) of a document to be allowed in the cache	
MCacheRemovalAlgorithm LRU GDSF	GDSF
The algorithm used to select documents for removal from the cache	
MCacheSize <i>KBytes</i>	100
The maximum amount of memory used by the cache in KBytes	
MergeTrailers [on off]	off
Determines whether trailers are merged into headers	
MetaDir <i>directory</i>	.web
Name of the directory to find CERN-style meta information files	
MetaFiles on off	off
Activates CERN meta-file processing	
MetaSuffix <i>suffix</i>	.meta
File name suffix for the file containing CERN-style meta information	
MimeMagicFile <i>file-path</i>	
Enable MIME-type determination based on file contents using the specified magic file	
MinSpareServers <i>Anzahl</i>	5
Minimale Anzahl der unbeschäftigten Kindprozesse des Servers	
MinSpareThreads <i>Anzahl</i>	
Minimale Anzahl unbeschäftigter Threads, die zur Bedienung von Anfragespitzen zur Verfügung	
MMapFile <i>file-path [file-path] ...</i>	
Map a list of files into memory at startup time	
ModMimeUsePathInfo On Off	Off
Tells <code>mod_mime</code> to treat <code>path_info</code> components as part of the filename	
MultiviewsMatch Any NegotiatedOnly Filters Handlers [Handlers Filters]	NegotiatedOnly
The types of files that will be included when searching for a matching file with MultiViews	
NameVirtualHost <i>Adresse[:Port]</i>	
Bestimmt eine IP-Adresse für den Betrieb namensbasierter virtueller Hosts	
NoProxy <i>host [host] ...</i>	
Hosts, domains, or networks that will be connected to directly	
NWSSLTrustedCerts <i>filename [filename] ...</i>	

List of additional client certificates	
NWSSLUpgradeable [<i>IP-address</i>]: <i>portnumber</i>	
Allows a connection to be upgraded to an SSL connection upon request	
Options [+] <i>Option</i> [-] <i>Option</i> ...	All
Definiert, welche Eigenschaften oder Funktionen in einem bestimmten Verzeichnis verfügbar sind	
Order <i>ordering</i>	Deny,Allow
Controls the default access state and the order in which Allow and Deny are evaluated.	
PassEnv <i>env-variable</i> [<i>env-variable</i>] ...	
Passes environment variables from the shell	
PidFile <i>Dateiname</i>	logs/httpd.pid
Datei, in welcher der Server die Prozess-ID des Daemons ablegt	
Protocol <i>protocol</i>	
Protocol for a listening socket	
ProtocolEcho On Off	Off
Turn the echo server on or off	
<Proxy <i>wildcard-url</i> > ...</Proxy>	
Container for directives applied to proxied resources	
ProxyBadHeader IsError Ignore StartBody	IsError
Determines how to handle bad header lines in a response	
ProxyBlock * <i>word</i> <i>host</i> <i>domain</i> [<i>word</i> <i>host</i> <i>domain</i>] ...	
Words, hosts, or domains that are banned from being proxied	
ProxyDomain <i>Domain</i>	
Default domain name for proxied requests	
ProxyErrorOverride On Off	Off
Override error pages for proxied content	
ProxyFtpDirCharset <i>character set</i>	ISO-8859-1
Define the character set for proxied FTP listings	
ProxyIOBufferSize <i>bytes</i>	8192
Determine size of internal data throughput buffer	
<ProxyMatch <i>regex</i> > ...</ProxyMatch>	
Container for directives applied to regular-expression-matched proxied resources	
ProxyMaxForwards <i>number</i>	-1
Maximum number of proxies that a request can be forwarded through	
ProxyPass [<i>path</i>] ! <i>url</i> [<i>key=value</i> [<i>key=value</i>	

...]] [nocanon] [interpolate]	
Maps remote servers into the local server URL-space	
ProxyPassInterpolateEnv On Off	Off
Enable Environment Variable interpolation in Reverse Proxy configurations	
ProxyPassMatch [regex] ! url [key=value [key=value ...]]	
Maps remote servers into the local server URL-space using regular expressions	
ProxyPassReverse [path] url [interpolate]	
Adjusts the URL in HTTP response headers sent from a reverse proxied server	
ProxyPassReverseCookieDomain internal-domain public-domain [interpolate]	
Adjusts the Domain string in Set-Cookie headers from a reverse- proxied server	
ProxyPassReverseCookiePath internal-path public-path [interpolate]	
Adjusts the Path string in Set-Cookie headers from a reverse- proxied server	
ProxyPreserveHost On Off	Off
Use incoming Host HTTP request header for proxy request	
ProxyReceiveBufferSize bytes	0
Network buffer size for proxied HTTP and FTP connections	
ProxyRemote match remote-server	
Remote proxy used to handle certain requests	
ProxyRemoteMatch regex remote-server	
Remote proxy used to handle requests matched by regular expressions	
ProxyRequests On Off	Off
Enables forward (standard) proxy requests	
ProxySCGIInternalRedirect On Off	On
Enable or disable internal redirect responses from the backend	
ProxySCGISendfile On Off Headername	Off
Enable evaluation of X-Sendfile pseudo response header	
ProxySet url key=value [key=value ...]	
Set various Proxy balancer or member parameters	
ProxyStatus Off On Full	Off
Show Proxy LoadBalancer status in mod_status	
ProxyTimeout seconds	
Network timeout for proxied requests	

ProxyVia On Off Full Block	Off
Information provided in the Via HTTP response header for proxied requests	
ReadmeName filename	
Name of the file that will be inserted at the end of the index listing	
ReceiveBufferSize bytes	0
TCP receive buffer size	
Redirect [status] URL-path URL	
Sends an external redirect asking the client to fetch a different URL	
RedirectMatch [status] regex URL	
Sends an external redirect based on a regular expression match of the current URL	
RedirectPermanent URL-path URL	
Sends an external permanent redirect asking the client to fetch a different URL	
RedirectTemp URL-path URL	
Sends an external temporary redirect asking the client to fetch a different URL	
RegisterHttpMethod method [method [...]]	
Register non-standard HTTP methods	
RemoveCharset extension [extension] ...	
Removes any character set associations for a set of file extensions	
RemoveEncoding extension [extension] ...	
Removes any content encoding associations for a set of file extensions	
RemoveHandler extension [extension] ...	
Removes any handler associations for a set of file extensions	
RemoveInputFilter extension [extension] ...	
Removes any input filter associations for a set of file extensions	
RemoveLanguage extension [extension] ...	
Removes any language associations for a set of file extensions	
RemoveOutputFilter extension [extension] ...	
Removes any output filter associations for a set of file extensions	
RemoveType extension [extension] ...	
Removes any content type associations for a set of file extensions	
RequestHeader	
set append merge add unset edit header [value] [replacement] [early env=!]variable	
Configure HTTP request headers	
RequestReadTimeout [header=timeout][-	

<u>maxtimeout</u> ,MinRate=rate] [<u>body=timeout</u> [[<u>maxtimeout</u>],MinRate=rate]	
Set timeout values for receiving request headers and body from client.	
<u>Require Name</u> [<u>Name</u>] ...	
Wählt die authentisierten Benutzer aus, die auf eine Ressource zugreifen können	
<u>RewriteBase</u> <u>URL-path</u>	
Sets the base URL for per-directory rewrites	
<u>RewriteCond</u> <u>TestString</u> <u>CondPattern</u>	
Defines a condition under which rewriting will take place	
<u>RewriteEngine</u> on off	off
Enables or disables runtime rewriting engine	
<u>RewriteLock</u> <u>file-path</u>	
Sets the name of the lock file used for <u>RewriteMap</u> synchronization	
<u>RewriteLog</u> <u>file-path</u> <u>pipe</u>	
Sets the name of the file used for logging rewrite engine processing	
<u>RewriteLogLevel</u> <u>Level</u>	0
Sets the verbosity of the log file used by the rewrite engine	
<u>RewriteMap</u> <u>MapName</u> <u>MapType</u> : <u>MapSource</u>	
Defines a mapping function for key-lookup	
<u>RewriteOptions</u> <u>Options</u>	
Sets some special options for the rewrite engine	
<u>RewriteRule</u> <u>Pattern</u> <u>Substitution</u> [<u>flags</u>]	
Defines rules for the rewriting engine	
<u>RLimitCPU</u> <u>Sekunden</u> max [<u>Sekunden</u> max]	
Begrenzt den CPU-Verbrauch von Prozessen, die von Apache-Kindprozessen gestartet wurden	
<u>RLimitMEM</u> <u>Bytes</u> max [<u>Bytes</u> max]	
Begrenzt den Speicherverbrauch von Prozessen, die von Apache-Kindprozessen gestartet wurden	
<u>RLimitNPROC</u> <u>Zahl</u> max [<u>Zahl</u> max]	
Begrenzt die Anzahl der Prozesse, die von Prozessen gestartet werden können, der ihre Apache-Kinprozessen gestartet wurden	
<u>Satisfy Any</u> All	All
Zusammenspiel von rechnerbasierter Zugriffskontrolle und Benutzerauthentisierung	
<u>ScoreBoardFile</u> <u>Dateipfad</u>	logs/apache_status
Ablageort der Datei, die zur Speicherung von Daten zur Koordinierung der Kindprozesse verwendet wird	
<u>Script</u> <u>Methode</u> <u>CGI-Skript</u>	

Aktiviert ein CGI-Skript für eine bestimmte Anfragemethode.	
<u>ScriptAlias</u> <i>URL-path file-path directory-path</i>	
Maps a URL to a filesystem location and designates the target as a CGI script	
<u>ScriptAliasMatch</u> <i>regex file-path directory-path</i>	
Maps a URL to a filesystem location using a regular expression and designates the target a	
<u>ScriptInterpreterSource</u> <i>Registry Registry-Strict Script</i>	Script
Methode zur Ermittlung des Interpreters von CGI-Skripten	
<u>ScriptLog</u> <i>file-path</i>	
Location of the CGI script error logfile	
<u>ScriptLogBuffer</u> <i>bytes</i>	1024
Maximum amount of PUT or POST requests that will be recorded in the scriptlog	
<u>ScriptLogLevel</u> <i>bytes</i>	10385760
Size limit of the CGI script logfile	
<u>ScriptSock</u> <i>file-path</i>	logs/cgisock
The filename prefix of the socket to use for communication with the cgi daemon	
<u>SecureListen</u> [<i>IP-address:</i>] <i>portnumber</i> <u>Certificate-Name</u> [<i>MUTUAL</i>]	
Enables SSL encryption for the specified port	
<u>SeeRequestTail</u> <i>On Off</i>	Off
Determine if mod_status displays the first 63 characters of a request or the last 63, assuming itself is greater than 63 chars.	
<u>SendBufferSize</u> <i>Bytes</i>	0
Größe des TCP-Puffers	
<u>ServerAdmin</u> <i>E-Mail-Adresse URL</i>	
E-Mail-Adresse, die der Server in Fehlermeldungen einfügt, welche an den Client gesendet	
<u>ServerAlias</u> <i>Hostname [Hostname] ...</i>	
Alternativer Name für einen Host, der verwendet wird, wenn Anfragen einem namensbasier Host zugeordnet werden	
<u>ServerLimit</u> <i>Anzahl</i>	
Obergrenze für die konfigurierbare Anzahl von Prozessen	
<u>ServerName</u> <i>voll-qualifizierter-Domainname[:port]</i>	
Rechnername und Port, die der Server dazu verwendet, sich selbst zu identifizieren	
<u>ServerPath</u> <i>URL-Pfad</i>	
Veralteter URL-Pfad für einen namensbasierten virtuellen Host, auf den von einem inkompa	

zugriffen wird	
ServerRoot <i>Verzeichnis</i>	/usr/local/apache
Basisverzeichnis der Serverinstallation	
ServerSignature On Off EMail	Off
Konfiguriert die Fußzeile von servergenerierten Dokumenten	
ServerTokens	Full
Major Minor Min[imal] Prod[uctOnly] OS Full	
Konfiguriert den HTTP-Response-Header Server	
SetEnv <i>env-variable</i> [<i>value</i>]	
Sets environment variables	
SetEnvIf <i>attribute regex</i> [!]<i>env-variable</i>[=<i>value</i>] [!]<i>env-variable</i>[=<i>value</i>] ...	
Sets environment variables based on attributes of the request	
SetEnvIfNoCase <i>attribute regex</i> [!]<i>env-variable</i>[=<i>value</i>] [!]<i>env-variable</i>[=<i>value</i>] ...	
Sets environment variables based on attributes of the request without respect to case	
SetHandler <i>Handlername</i> None	
Erzwingt die Verarbeitung aller passenden Dateien durch einen Handler	
SetInputFilter <i>Filter</i>[:<i>Filter</i>...]	
Bestimmt die Filter, die Client-Anfragen und POST-Eingaben verarbeiten	
SetOutputFilter <i>Filter</i>[:<i>Filter</i>...]	
Bestimmt die Filter, die Antworten des Servers verarbeiten	
SSIEnableAccess on off	off
Enable the -A flag during conditional flow control processing.	
SSIEndTag <i>tag</i>	"-->"
String that ends an include element	
SSIErrorMsg <i>message</i>	"[an error occurred +
Error message displayed when there is an SSI error	
SSIETag on off	off
Controls whether ETags are generated by the server.	
SSILastModified on off	off
Controls whether Last-Modified headers are generated by the server.	
SSIStartTag <i>tag</i>	"<!--#"
String that starts an include element	
SSITimeFormat <i>formatstring</i>	"%A, %d-%b-%Y

	%H:%M +
Configures the format in which date strings are displayed	
<u>SSIUndefinedEcho</u> <i>string</i>	"(none)"
String displayed when an unset variable is echoed	
<u>SSLCACertificateFile</u> <i>file-path</i>	
File of concatenated PEM-encoded CA Certificates for Client Auth	
<u>SSLCACertificatePath</u> <i>directory-path</i>	
Directory of PEM-encoded CA Certificates for Client Auth	
<u>SSLCADNRequestFile</u> <i>file-path</i>	
File of concatenated PEM-encoded CA Certificates for defining acceptable CA names	
<u>SSLCADNRequestPath</u> <i>directory-path</i>	
Directory of PEM-encoded CA Certificates for defining acceptable CA names	
<u>SSLCARevocationFile</u> <i>file-path</i>	
File of concatenated PEM-encoded CA CRLs for Client Auth	
<u>SSLCARevocationPath</u> <i>directory-path</i>	
Directory of PEM-encoded CA CRLs for Client Auth	
<u>SSLCertificateChainFile</u> <i>file-path</i>	
File of PEM-encoded Server CA Certificates	
<u>SSLCertificateFile</u> <i>file-path</i>	
Server PEM-encoded X.509 Certificate file	
<u>SSLCertificateKeyFile</u> <i>file-path</i>	
Server PEM-encoded Private Key file	
<u>SSLCipherSuite</u> <i>cipher-spec</i>	ALL:!ADH:RC4+RSA: +
Cipher Suite available for negotiation in SSL handshake	
<u>SSLCompression</u> <i>on off</i>	off
Enable compression on the SSL level	
<u>SSLCryptoDevice</u> <i>engine</i>	builtin
Enable use of a cryptographic hardware accelerator	
<u>SSLEngine</u> <i>on off optional</i>	off
SSL Engine Operation Switch	
<u>SSLFIPS</u> <i>on off</i>	off
SSL FIPS mode Switch	
<u>SSLHonorCipherOrder</u> <i>flag</i>	
Option to prefer the server's cipher preference order	

<u>SSLInsecureRenegotiation <i>flag</i></u>	off
Option to enable support for insecure renegotiation	
<u>SSLMutex <i>type</i></u>	none
Semaphore for internal mutual exclusion of operations	
<u>SSLOptions [+ -]<i>option ...</i></u>	
Configure various SSL engine run-time options	
<u>SSLPassPhraseDialog <i>type</i></u>	builtin
Type of pass phrase dialog for encrypted private keys	
<u>SSLProtocol [+ -]<i>protocol ...</i></u>	all
Configure usable SSL protocol flavors	
<u>SSLProxyCACertificateFile <i>file-path</i></u>	
File of concatenated PEM-encoded CA Certificates for Remote Server Auth	
<u>SSLProxyCACertificatePath <i>directory-path</i></u>	
Directory of PEM-encoded CA Certificates for Remote Server Auth	
<u>SSLProxyCARevocationFile <i>file-path</i></u>	
File of concatenated PEM-encoded CA CRLs for Remote Server Auth	
<u>SSLProxyCARevocationPath <i>directory-path</i></u>	
Directory of PEM-encoded CA CRLs for Remote Server Auth	
<u>SSLProxyCheckPeerCN on off</u>	off
Whether to check the remote server certificates CN field	
<u>SSLProxyCheckPeerExpire on off</u>	off
Whether to check if remote server certificate is expired	
<u>SSLProxyCipherSuite <i>cipher-spec</i></u>	ALL:!ADH:RC4+RSA: +
Cipher Suite available for negotiation in SSL proxy handshake	
<u>SSLProxyEngine on off</u>	off
SSL Proxy Engine Operation Switch	
<u>SSLProxyMachineCertificateChainFile <i>filename</i></u>	
File of concatenated PEM-encoded CA certificates to be used by the proxy for choosing a c	
<u>SSLProxyMachineCertificateFile <i>filename</i></u>	
File of concatenated PEM-encoded client certificates and keys to be used by the proxy	
<u>SSLProxyMachineCertificatePath <i>directory</i></u>	
Directory of PEM-encoded client certificates and keys to be used by the proxy	
<u>SSLProxyProtocol [+ -]<i>protocol ...</i></u>	all

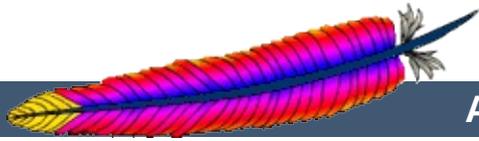
Configure usable SSL protocol flavors for proxy usage	
<u>SSLProxyVerify <i>level</i></u>	none
Type of remote server Certificate verification	
<u>SSLProxyVerifyDepth <i>number</i></u>	1
Maximum depth of CA Certificates in Remote Server Certificate verification	
<u>SSLRandomSeed <i>context source [bytes]</i></u>	
Pseudo Random Number Generator (PRNG) seeding source	
<u>SSLRenegBufferSize <i>bytes</i></u>	131072
Set the size for the SSL renegotiation buffer	
<u>SSLRequire <i>expression</i></u>	
Allow access only when an arbitrarily complex boolean expression is true	
<u>SSLRequireSSL</u>	
Deny access when SSL is not used for the HTTP request	
<u>SSLSessionCache <i>type</i></u>	none
Type of the global/inter-process SSL Session Cache	
<u>SSLSessionCacheTimeout <i>seconds</i></u>	300
Number of seconds before an SSL session expires in the Session Cache	
<u>SSLSessionTicketKeyFile <i>file-path</i></u>	
Persistent encryption/decryption key for TLS session tickets	
<u>SSLSessionTickets <i>on off</i></u>	on
Enable or disable use of TLS session tickets	
<u>SSLStrictSNIVHostCheck <i>on off</i></u>	off
Whether to allow non SNI clients to access a name based virtual host.	
<u>SSLUserName <i>varname</i></u>	
Variable name to determine user name	
<u>SSLVerifyClient <i>level</i></u>	none
Type of Client Certificate verification	
<u>SSLVerifyDepth <i>number</i></u>	1
Maximum depth of CA Certificates in Client Certificate verification	
<u>StartServers <i>Anzahl</i></u>	
Anzahl der Kindprozesse des Servers, die beim Start erstellt werden	
<u>StartThreads <i>Anzahl</i></u>	
Anzahl der Threads, die beim Start erstellt werden	
<u>Substitute <i>s/pattern/substitution/[infq]</i></u>	
Pattern to filter the response content	

SubstituteInheritBefore <i>on off</i>	off
Change the merge order of inherited patterns	
Suexec <i>On Off</i>	
Enable or disable the suEXEC feature	
SuexecUserGroup <i>User Group</i>	
User and group for CGI programs to run as	
ThreadLimit <i>Anzahl</i>	
Bestimmt die Obergrenze der konfigurierbaren Anzahl von Threads pro Kindprozess	
ThreadsPerChild <i>Anzahl</i>	
Anzahl der Threads, die mit jedem Kindprozess gestartet werden	
ThreadStackSize <i>size</i>	
Die Größe des Stacks in Bytes, der von Threads verwendet wird, die Client-Verbindungen b	
TimeOut <i>Sekunden</i>	300
Zeitspanne, die der Server auf verschiedene Ereignisse wartet, bevor er die Anfrage abbrich	
TraceEnable <i>[on off extended]</i>	on
Determines the behaviour on TRACE requests	
TransferLog <i>file pipe</i>	
Specify location of a log file	
TypesConfig <i>file-path</i>	conf/mime.types
The location of the mime . types file	
UnsetEnv <i>env-variable [env-variable] ...</i>	
Removes variables from the environment	
UseCanonicalName <i>On Off DNS</i>	Off
Bestimmt, wie der Server seinen eigenen Namen und Port ermittelt	
UseCanonicalPhysicalPort <i>On Off</i>	Off
Configures how the server determines its own name and port	
User <i>Unix-User-ID</i>	#-1
Die Benutzerkennung, unter welcher der Server Anfragen beantwortet	
UserDir <i>directory-filename [directory-filename] ...</i>	
Location of the user-specific directories	
VirtualDocumentRoot <i>interpolated-directory none</i>	none
Dynamically configure the location of the document root for a given virtual host	
VirtualDocumentRootIP <i>interpolated-</i>	none

<u>directory none</u>	
Dynamically configure the location of the document root for a given virtual host	
<u><VirtualHost Adresse[:Port] [Adresse[:Port]]</u>	
<u>...> ... </VirtualHost></u>	
Enthält Direktiven, die nur auf bestimmte Hostnamen oder IP-Adressen angewendet werden	
<u>VirtualScriptAlias interpolated-directory none</u>	none
Dynamically configure the location of the CGI directory for a given virtual host	
<u>VirtualScriptAliasIP interpolated-</u>	none
<u>directory none</u>	
Dynamically configure the location of the cgi directory for a given virtual host	
<u>Win32DisableAcceptEx</u>	
Für die Annahme von Netzwerkverbindungen wird accept() anstelle von AcceptEx() verwendet	
<u>XBitHack on off full</u>	off
Parse SSI directives in files with the execute bit set	

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Modul-Index

Unten ist eine Liste aller Module angegeben, die als Bestandteil der Apache-Distribution mitgeliefert werden. Bitte beachten Sie auch die vollständige alphabetische Liste [aller Apache-Direktiven](#).

Siehe auch

[Multi-Processing-Module \(MPMs\)](#)

[Kurzreferenz der Direktiven](#)



core

Ständig verfügbare Kernfunktionen des Apache HTTP Servers

mpm_common

Eine Sammlung von Direktiven, die in mehr als einem Multi-Processing-Modul (MPM) implementiert sind.

beos

Dieses Multi-Processing-Modul ist für BeOS optimiert.

event

An experimental variant of the standard worker MPM

mpm_netware

Multi-Processing Module implementing an exclusively threaded web server optimized for Novell NetWare

mpmt_os2

Hybrid multi-process, multi-threaded MPM for OS/2

prefork

Implementiert einen im Voraus forkenden Webserver ohne Thread-Unterstützung

mpm_winnt

Das Multi-Processing-Modul ist optimiert für Windows NT.

worker

Multi-Processing-Modul, das einen Hybrid-Webserver mit Multi-Thread und Multi-Prozess-Unterstützung implementiert



A | C | D | E | F | H | I | L | M | N | P | R | S | U | V

[mod_actions](#)

Dieses Modul ermöglicht die Ausführung von CGI-Skripten in Abhängigkeit von Medientypen und Anfragemethoden.

[mod_alias](#)

Provides for mapping different parts of the host filesystem in the document tree and for URL redirection

[mod_asis](#)

Sends files that contain their own HTTP headers

[mod_auth_basic](#)

Basic authentication

[mod_auth_digest](#)

User authentication using MD5 Digest Authentication.

[mod_authn_alias](#)

Provides the ability to create extended authentication providers based on actual providers

[mod_authn_anon](#)

Allows "anonymous" user access to authenticated areas

[mod_authn_dbd](#)

User authentication using an SQL database

[mod_authn_dbm](#)

User authentication using DBM files

[mod_authn_default](#)

Authentication fallback module

[mod_authn_file](#)

User authentication using text files

[mod_authnz_ldap](#)

Allows an LDAP directory to be used to store the database for HTTP

Basic authentication.

[mod_authz_dbm](#)

Group authorization using DBM files

[mod_authz_default](#)

Authorization fallback module

[mod_authz_groupfile](#)

Group authorization using plaintext files

[mod_authz_host](#)

Group authorizations based on host (name or IP address)

[mod_authz_owner](#)

Authorization based on file ownership

[mod_authz_user](#)

User Authorization

[mod_autoindex](#)

Generates directory indexes, automatically, similar to the Unix `ls` command or the Win32 `dir` shell command

[mod_cache](#)

Content cache keyed to URIs.

[mod_cern_meta](#)

CERN httpd metafile semantics

[mod_cgi](#)

Execution of CGI scripts

[mod_cgid](#)

Execution of CGI scripts using an external CGI daemon

[mod_charset_lite](#)

Specify character set translation or recoding

[mod_dav](#)

Distributed Authoring and Versioning ([WebDAV](#)) functionality

[mod_dav_fs](#)

filesystem provider for [mod_dav](#)

[mod_dav_lock](#)

generic locking module for [mod_dav](#)

[mod_dbd](#)

Manages SQL database connections

[mod_deflate](#)

Compress content before it is delivered to the client

[mod_dir](#)

Provides for "trailing slash" redirects and serving directory index files

[mod_disk_cache](#)

Content cache storage manager keyed to URIs

[mod_dumpio](#)

Dumps all I/O to error log as desired.

[mod_echo](#)

A simple echo server to illustrate protocol modules

[mod_env](#)

Modifies the environment which is passed to CGI scripts and SSI pages

[mod_example](#)

Illustrates the Apache module API

[mod_expires](#)

Generation of Expires and Cache-Control HTTP headers according to user-specified criteria

[mod_ext_filter](#)

Pass the response body through an external program before delivery to the client

[mod_file_cache](#)

Caches a static list of files in memory

[mod_filter](#)

Context-sensitive smart filter configuration module

[mod_headers](#)

Customization of HTTP request and response headers

[mod_ident](#)

RFC 1413 ident lookups

[mod_imagemap](#)

Server-side imagemap processing

[mod_include](#)

Server-parsed html documents (Server Side Includes)

[mod_info](#)

Provides a comprehensive overview of the server configuration

[mod_isapi](#)

ISAPI Extensions within Apache for Windows

[mod_ldap](#)

LDAP connection pooling and result caching services for use by other LDAP modules

[mod_log_config](#)

Logging of the requests made to the server

[mod_log_forensic](#)

Forensic Logging of the requests made to the server

[mod_logio](#)

Logging of input and output bytes per request

[mod_mem_cache](#)

Content cache keyed to URIs

[mod_mime](#)

Associates the requested filename's extensions with the file's behavior (handlers and filters) and content (mime-type, language, character set and encoding)

[mod_mime_magic](#)

Determines the MIME type of a file by looking at a few bytes of its contents

[mod_negotiation](#)

Provides for [content negotiation](#)

[mod_nw_ssl](#)

Enable SSL encryption for NetWare

[mod_proxy](#)

HTTP/1.1 proxy/gateway server

[mod_proxy_ajp](#)

AJP support module for [mod_proxy](#)

[mod_proxy_balancer](#)

[mod_proxy](#) extension for load balancing

[mod_proxy_connect](#)

[mod_proxy](#) extension for CONNECT request handling

[mod_proxy_ftp](#)

FTP support module for [mod_proxy](#)

[mod_proxy_http](#)

HTTP support module for [mod_proxy](#)

[mod_proxy_scgi](#)

SCGI gateway module for [mod_proxy](#)

[mod_reqtimeout](#)

Set timeout and minimum data rate for receiving requests

[mod_rewrite](#)

Provides a rule-based rewriting engine to rewrite requested URLs on the fly

[mod_setenvif](#)

Allows the setting of environment variables based on characteristics of the request

[mod_so](#)

Loading of executable code and modules into the server at start-up or restart time

mod_speling

Attempts to correct mistaken URLs that users might have entered by ignoring capitalization and by allowing up to one misspelling

mod_ssl

Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols

mod_status

Provides information on server activity and performance

mod_substitute

Perform search and replace operations on response bodies

mod_suexec

Allows CGI scripts to run as a specified user and Group

mod_unique_id

Provides an environment variable with a unique identifier for each request

mod_userdir

User-specific directories

mod_usertrack

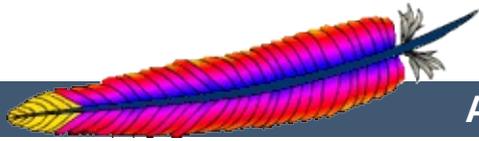
Clickstream logging of user activity on a site

mod_version

Version dependent configuration

mod_vhost_alias

Provides for dynamically configured mass virtual hosting



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Seitenindex

Diese Seite verzeichnet die zur Zeit verfügbaren Dokumente der [Dokumentation zum Apache HTTP Server Version 2.2.](#)



-
- [Upgrade von 1.3 auf 2.0](#)
 - [Neue Funktionen in Version 2.1/2.2](#)
 - [Neue Funktionen in Version 2.0](#)
 - [Apache-Lizenz](#)



- [Kompilieren und Installieren](#)
- [Apache starten](#)
- [Beenden und Neustarten des Servers](#)
- [Konfigurationsdateien](#)
- [Konfigurationsabschnitte](#)
- [Caching von Inhalten](#)
- [Serverweite Konfiguration](#)
- [Log-Dateien](#)
- [URLs auf das Dateisystem abbilden](#)
- [Dynamic Shared Object \(DSO\)](#)
- [Content Negotiation](#)
- [Individuelle Fehlermeldungen](#)
- [Bestimmen der vom Apache verwendeten Adressen und Ports](#)
- [Multi-Processing-Module \(MPMs\)](#)
- [Umgebungsvariablen](#)
- [Handler](#)
- [Filter](#)
- [suEXEC Unterstützung](#)
- [Probleme bezüglich DNS und Apache](#)
- [Häufig gestellte Fragen \(FAQ\)](#)



-
- [Übersicht](#)
 - [Namensbasierte virtuelle Hosts](#)
 - [IP-basierte virtuelle Hosts](#)
 - [Dynamisch konfiguriertes Massen-Virtual-Hosting](#)
 - [Beispiele für virtuelle Hosts in typischen Installationen](#)
 - [Tiefergehende Erörterung der Zuweisung virtueller Hosts](#)
 - [Datei-Deskriptor-Begrenzungen](#)



-
- [Übersicht](#)
 - [Referenz-Dokumentation von mod_rewrite](#)
 - [Einführung in reguläre Ausdrücke und mod_rewrite](#)
 - [Verwendung von mod_rewrite, um URLs umzuleiten oder umzuschreiben](#)
 - [Zugriffskontrolle mit mod_rewrite](#)
 - [Dynamische virtuelle Hosts mit mod_rewrite](#)
 - [Dynamische Proxy-Konfigurationen mit mod_rewrite](#)
 - [RewriteMap einsetzen](#)
 - [Fortgeschrittene Techniken](#)
 - [Wann man auf mod_rewrite verzichten sollte](#)
 - [Schalter / Flags](#)
 - [Technische Details](#)



- [Übersicht](#)
- [SSL/TLS-Verschlüsselung: Einführung](#)
- [SSL/TLS-Verschlüsselung: Kompatibilität](#)
- [SSL/TLS-Verschlüsselung: Praxis](#)
- [SSL/TLS-Verschlüsselung: FAQ](#)



- [Übersicht](#)
- [Authentisierung und Autorisierung](#)
- [Zugriffskontrolle](#)
- [Dynamische Inhalte mit CGI](#)
- [Einführung in Server Side Includes](#)
- [.htaccess-Dateien](#)
- [Web-Verzeichnisse für Benutzer](#)



- [Übersicht](#)
- [Apache unter Microsoft Windows einsetzen](#)
- [Kompilieren des Apache für Microsoft Windows](#)
- [Apache unter Novell NetWare einsetzen](#)
- [Einen Hochleistungs-Web-Server auf HPUX betreiben](#)
- [Die Apache EBCDIC-Portierung](#)



- [Übersicht](#)
- [httpd](#)
- [ab](#)
- [apachectl](#)
- [apxs](#)
- [configure](#)
- [dbmmanage](#)
- [htcacheclean](#)
- [htdbm](#)
- [htdigest](#)
- [htpasswd](#)
- [htt2dbm](#)
- [logresolve](#)
- [rotatelogs](#)
- [suexec](#)
- [Sonstige Programme](#)



- [Übersicht](#)
- [Performance-Hinweise](#)
- [Tipps zur Sicherheit](#)
- [Wichtige Standards](#)
- [Verschlüsselungsformate für Passworte](#)



- [Erklärung der Fachbegriffe zu Apache-Modulen](#)
- [Erklärung der Fachbegriffe zu Apache-Direktiven](#)

- [Apache-Kernfunktionen](#)
- [Allgemeine Direktiven der Apache-MPMs](#)
- [Apache-MPM beos](#)
- [Apache-MPM event](#)
- [Apache-MPM netware](#)
- [Apache-MPM os2](#)
- [Apache-MPM prefork](#)
- [Apache-MPM winnt](#)
- [Apache-MPM worker](#)

- [Apache-Modul mod_actions](#)
- [Apache-Modul mod_alias](#)
- [Apache-Modul mod_asis](#)
- [Apache-Modul mod_auth_basic](#)
- [Apache-Modul mod_auth_digest](#)
- [Apache-Modul mod_authn_alias](#)
- [Apache-Modul mod_authn_anon](#)
- [Apache-Modul mod_authn_dbd](#)
- [Apache-Modul mod_authn_dbm](#)
- [Apache-Modul mod_authn_default](#)
- [Apache-Modul mod_authn_file](#)
- [Apache-Modul mod_authnz_ldap](#)
- [Apache-Modul mod_authz_dbm](#)
- [Apache-Modul mod_authz_default](#)
- [Apache-Modul mod_authz_groupfile](#)
- [Apache-Modul mod_authz_host](#)
- [Apache-Modul mod_authz_owner](#)
- [Apache-Modul mod_authz_user](#)
- [Apache-Modul mod_autoindex](#)
- [Apache-Modul mod_cache](#)

- [Apache-Modul mod_cern_meta](#)
- [Apache-Modul mod_cgi](#)
- [Apache-Modul mod_cgid](#)
- [Apache-Modul mod_charset_lite](#)
- [Apache-Modul mod_dav](#)
- [Apache-Modul mod_dav_fs](#)
- [Apache-Modul mod_dav_lock](#)
- [Apache-Modul mod_dbd](#)
- [Apache-Modul mod_deflate](#)
- [Apache-Modul mod_dir](#)
- [Apache-Modul mod_disk_cache](#)
- [Apache-Modul mod_dumpio](#)
- [Apache-Modul mod_echo](#)
- [Apache-Modul mod_env](#)
- [Apache-Modul mod_example](#)
- [Apache-Modul mod_expires](#)
- [Apache-Modul mod_ext_filter](#)
- [Apache-Modul mod_file_cache](#)
- [Apache-Modul mod_filter](#)
- [Apache-Modul mod_headers](#)
- [Apache-Modul mod_ident](#)
- [Apache-Modul mod_imagemap](#)
- [Apache-Modul mod_include](#)
- [Apache-Modul mod_info](#)
- [Apache-Modul mod_isapi](#)
- [Apache-Modul mod_ldap](#)
- [Apache-Modul mod_log_config](#)
- [Apache-Modul mod_log_forensic](#)
- [Apache-Modul mod_logio](#)
- [Apache-Modul mod_mem_cache](#)
- [Apache-Modul mod_mime](#)
- [Apache-Modul mod_mime_magic](#)
- [Apache-Modul mod_negotiation](#)
- [Apache-Modul mod_nw_ssl](#)

- [Apache-Modul mod_proxy](#)
- [Apache-Modul mod_proxy_ajp](#)
- [Apache-Modul mod_proxy_balancer](#)
- [Apache-Modul mod_proxy_connect](#)
- [Apache-Modul mod_proxy_ftp](#)
- [Apache-Modul mod_proxy_http](#)
- [Apache-Modul mod_proxy_scgi](#)
- [Apache-Modul mod_reqtimeout](#)
- [Apache-Modul mod_rewrite](#)
- [Apache-Modul mod_setenvif](#)
- [Apache-Modul mod_so](#)
- [Apache-Modul mod_speling](#)
- [Apache-Modul mod_ssl](#)
- [Apache-Modul mod_status](#)
- [Apache-Modul mod_substitute](#)
- [Apache-Modul mod_suexec](#)
- [Apache-Modul mod_unique_id](#)
- [Apache-Modul mod_userdir](#)
- [Apache-Modul mod_usertrack](#)
- [Apache-Modul mod_version](#)
- [Apache-Modul mod_vhost_alias](#)



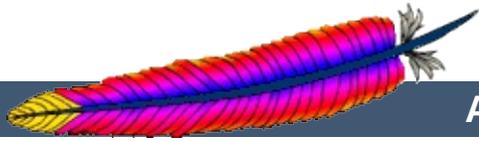
-
- [Übersicht](#)
 - [Anmerkungen zur Apache-API](#)
 - [Debuggen der Speicher-Belegung in der APR](#)
 - [Apache 2.0 dokumentieren](#)
 - [Hook-Funktionen des Apache 2.0](#)
 - [Module von Apache 1.3 nach Apache 2.0 konvertieren](#)
 - [Verarbeitung der Anfragen im Apache 2.0](#)
 - [Wie Filter im Apache 2.0 arbeiten](#)
 - [Thread-Sicherheit in Version 2.0](#)



- [Glossar](#)
- [Modul-Index](#)
- [Direktiven-Index](#)
- [Kurzreferenz der Direktiven](#)

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Server and Supporting Programs

This page documents all the executable programs included with the Apache HTTP Server.



[httpd](#)

Apache hypertext transfer protocol server

[apachectl](#)

Apache HTTP server control interface

[ab](#)

Apache HTTP server benchmarking tool

[apxs](#)

APache eXtenSion tool

[configure](#)

Configure the source tree

[dbmmanage](#)

Create and update user authentication files in DBM format for basic authentication

[htcacheclean](#)

Clean up the disk cache

[htdigest](#)

Create and update user authentication files for digest authentication

[htdbm](#)

Manipulate DBM password databases.

[htpasswd](#)

Create and update user authentication files for basic authentication

[htt2dbm](#)

Create dbm files for use with RewriteMap

[logresolve](#)

Resolve hostnames for IP-addresses in Apache logfiles

[rotatelogs](#)

Rotate Apache logs without having to kill the server

[suexec](#)

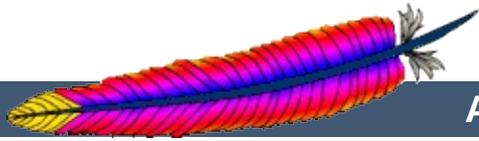
Switch User For Exec

Other Programs

Support tools with no own manual page.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Apache SSL/TLS Encryption

The Apache HTTP Server module `mod_ssl` provides an interface to the [OpenSSL](#) library, which provides Strong Encryption using the Secure Sockets Layer and Transport Layer Security protocols. The module and this documentation are based on Ralf S. Engelschall's `mod_ssl` project.



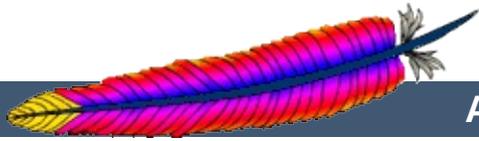
-
- [Introduction](#)
 - [Compatibility](#)
 - [How-To](#)
 - [Frequently Asked Questions](#)
 - [Glossary](#)



Extensive documentation on the directives and environment variables provided by this module is provided in the [mod_ssl reference documentation](#).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Apache mod_rewrite

[mod_rewrite](#) provides a way to modify incoming URL requests, dynamically, based on [regular expression](#) rules. This allows you to map arbitrary URLs onto your internal URL structure in any way you like.

It supports an unlimited number of rules and an unlimited number of attached rule conditions for each rule to provide a really flexible and powerful URL manipulation mechanism. The URL manipulations can depend on various tests: server variables, environment variables, HTTP headers, time stamps, external database lookups, and various other external programs or handlers, can be used to achieve granular URL matching.

Rewrite rules can operate on the full URLs, including the path-info and query string portions, and may be used in per-server context (`httpd.conf`), per-virtualhost context (`<VirtualHost>` blocks), or per-directory context (`.htaccess` files and `<Directory>` blocks). The rewritten result can lead to further rules, internal sub-processing, external request redirection, or proxy passthrough, depending on what [flags](#) you attach to the rules.

Since `mod_rewrite` is so powerful, it can indeed be rather complex. This document supplements the [reference documentation](#), and attempts to allay some of that complexity, and provide highly annotated examples of common scenarios that you may handle with `mod_rewrite`. But we also attempt to show you when you should not use `mod_rewrite`, and use other standard Apache features instead, thus avoiding this unnecessary complexity.

- [mod_rewrite reference documentation](#)
- [Introduction to regular expressions and mod_rewrite](#)
- [Using mod_rewrite for redirection and remapping of URLs](#)

- [Using mod_rewrite to control access](#)
- [Dynamic virtual hosts with mod_rewrite](#)
- [Dynamic proxying with mod_rewrite](#)
- [Using RewriteMap](#)
- [Advanced techniques and tricks](#)
- [When **NOT** to use mod_rewrite](#)
- [RewriteRule Flags](#)
- [Technical details](#)

See also

[mod_rewrite reference documentation](#)

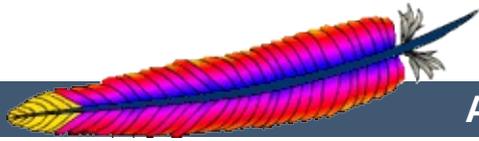
[Mapping URLs to the Filesystem](#)

[mod_rewrite wiki](#)

[Glossary](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP-Server](#) > [Dokumentation](#) > [Version 2.2](#)

Apache-Dokumentation zu virtuellen Hosts

Diese Übersetzung ist möglicherweise nicht mehr aktuell. Bitte prüfen Sie die englische Version auf die neuesten Änderungen.

Der Begriff *virtueller Host* (*Anm.d.Ü.:* engl. 'virtual host') bezieht sich auf die Praxis, mehr als ein Webangebot (z.B. `www.company1.com` und `www.company2.com`) auf einer einzigen Maschine zu betreiben. Virtuelle Hosts können "[IP-basiert](#)" sein, was bedeutet, dass jedes Webangebot eine andere IP besitzt, oder "[Namens-basiert](#)", was bedeutet, dass unter jeder IP-Adresse mehrere Namen laufen. Die Tatsache, dass sie auf dem gleichen physischen Server laufen, ist für den Endbenutzer nicht offensichtlich.

Der Apache war einer der ersten Server, der IP-basierte virtuelle Hosts von Haus aus direkt unterstützt hat. Seit Version 1.1 unterstützt der Apache sowohl IP-basierte als auch namensbasierte virtuelle Hosts (vhosts). Letzteres wird zuweilen auch *Host-basiert* oder *non-IP-Virtual-Host* genannt.

Nachfolgend finden Sie eine Liste von Dokumenten, die alle Details der Unterstützung von virtuellen Hosts ab Apache Version 1.3 beschreiben.

Siehe auch

[mod_vhost_alias](#)

[Namensbasierte virtuelle Hosts](#)

[IP-basierte virtuelle Hosts](#)

[Beispiele für virtuelle Hosts](#)

[Datei-Deskriptor-Begrenzungen](#)

[Massen-Virtual-Hosting](#)

[Zuweisung virtueller Hosts](#)



- [Namensbasierte virtuelle Hosts](#) (Mehr als ein Webangebot pro IP-Adresse)
- [IP-basierte virtuelle Hosts](#) (Eine IP-Adresse für jedes Webangebot)
- [Beispiele für virtuelles Hosts in typischen Installationen](#)
- [Datei-Deskriptor-Begrenzungen](#) (oder *Zu viele Protokolldateien*)
- [Dynamisch konfiguriertes Massen-Virtual-Hosting](#)
- [Tiefergehende Erörterung der Zuweisung virtueller Hosts](#)



- [<VirtualHost>](#)
- [NameVirtualHost](#)
- [ServerName](#)
- [ServerAlias](#)
- [ServerPath](#)

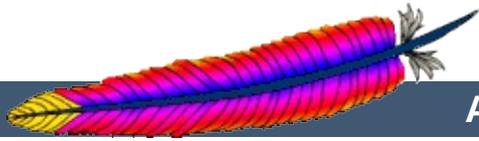
Bei der Suche von Fehlern in Ihrer Virtual-Host-Konfiguration ist die Apache-Befehlszeilenoption `-S` möglicherweise hilfreich. Geben Sie dazu den folgenden Befehl ein:

```
/usr/local/apache2/bin/httpd -S
```

Diese Anweisung gibt eine Beschreibung aus, wie der Apache die Konfigurationsdatei analysiert hat. Eine sorgfältige Überprüfung der IP-Adressen und Servernamen kann helfen, Konfigurationsfehler aufzudecken. (Lesen Sie die Dokumentation zum [httpd](#)-Programm für weitere Befehlszeilenoptionen.)

Copyright 2017 The Apache Software Foundation.
Lizenziert unter der [Apache License, Version 2.0](#).

[Module](#) | [Direktiven](#) | [FAQ](#) | [Glossar](#) | [Seitenindex](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Developer Documentation for Apache 2.x

Warning

Many of the documents listed here are in need of update. They are in different stages of progress. Please be patient, and point out any discrepancies or errors on the developer/ pages directly to the dev@httpd.apache.org mailing list.



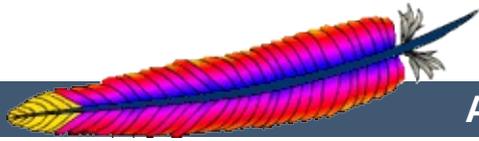
-
- [Apache 1.3 API Notes](#)
 - [Apache 2.0 Hook Functions](#)
 - [Request Processing in Apache 2.0](#)
 - [How filters work in Apache 2.0](#)
 - [Converting Modules from Apache 1.3 to Apache 2.0](#)
 - [Documenting Apache 2.0](#)
 - [Apache 2.x Thread Safety Issues](#)



-
- Developer articles at [apachetutor](#) include:
 - [Request Processing in Apache](#)
 - [Configuration for Modules](#)
 - [Resource Management in Apache](#)
 - [Connection Pooling in Apache](#)
 - [Introduction to Buckets and Brigades](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Apache Miscellaneous Documentation

Below is a list of additional documentation pages that apply to the Apache web server development project.

Warning

The documents below have not been fully updated to take into account changes made in the 2.1 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

[Performance Notes - Apache Tuning](#)

Notes about how to (run-time and compile-time) configure Apache for highest performance. Notes explaining why Apache does some things, and why it doesn't do other things (which make it slower/faster).

[Security Tips](#)

Some "do"s - and "don't"s - for keeping your Apache web site secure.

[URL Rewriting Guide](#)

This document supplements the [mod_rewrite reference documentation](#). It describes how one can use Apache's [mod_rewrite](#) to solve typical URL-based problems webmasters are usually confronted with in practice.

[Relevant Standards](#)

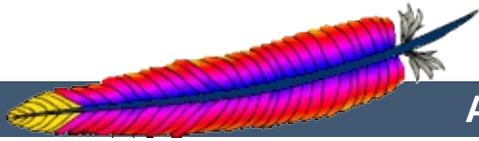
This document acts as a reference page for most of the relevant standards that Apache follows.

Password Encryption Formats

Discussion of the various ciphers supported by Apache for authentication purposes.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Platform Specific Notes



Using Apache

This document explains how to install, configure and run Apache 2.0 under Microsoft Windows.

See: [Using Apache with Microsoft Windows](#)

Compiling Apache

There are many important points before you begin compiling Apache. This document explain them.

See: [Compiling Apache for Microsoft Windows](#)



Novell NetWare

This document explains how to install, configure and run Apache 2.0 under Novell NetWare 5.1 and above.

See: [Using Apache With Novell NetWare](#)

EBCDIC

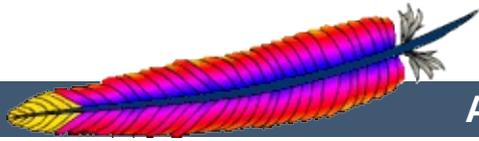
Version 1.3 of the Apache HTTP Server is the first version which includes a port to a (non-ASCII) mainframe machine which uses the EBCDIC character set as its native codeset.

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

See: [The Apache EBCDIC Port](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

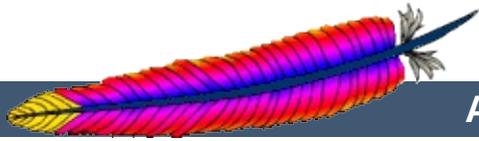
[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

Frequently Asked Questions

The FAQ has moved to the [HTTP Server Wiki](#).

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#) > [Programs](#)

suexec - Switch user before executing external programs

suexec is used by the Apache HTTP Server to switch to another user before executing CGI programs. In order to achieve this, it must run as root. Since the HTTP daemon normally doesn't run as root, the suexec executable needs the setuid bit set and must be owned by root. It should never be writable for any other person than root.

For further information about the concepts and the security model of suexec please refer to the suexec documentation (<http://httpd.apache.org/docs/2.2/suexec.html>).



suexec -V

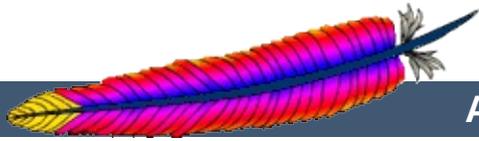


-V

If you are root, this option displays the compile options of suexec. For security reasons all configuration options are changeable only at compile time.

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)



Apache HTTP Server Version 2.2

[Apache](#) > [HTTP Server](#) > [Documentation](#) > [Version 2.2](#)

How-To / Tutorials



Authentication and Authorization

Authentication is any process by which you verify that someone is who they claim they are. Authorization is any process by which someone is allowed to be where they want to go, or to have information that they want to have.

See: [Authentication, Authorization](#)

Access Control

Access control refers to the process of restricting, or granting access to a resource based on arbitrary criteria. There are a variety of different ways that this can be accomplished.

See: [Access Control](#)

Dynamic Content with CGI

The CGI (Common Gateway Interface) defines a way for a web server to interact with external content-generating programs, which are often referred to as CGI programs or CGI scripts. It is the simplest, and most common, way to put dynamic content on your web site. This document will be an introduction to setting up CGI on your Apache web server, and getting started writing CGI programs.

See: [CGI: Dynamic Content](#)

.htaccess files

.htaccess files provide a way to make configuration changes on a per-directory basis. A file, containing one or more configuration directives, is placed in a particular document directory, and the directives apply to that directory, and all subdirectories thereof.

See: [.htaccess files](#)

Introduction to Server Side Includes

SSI (Server Side Includes) are directives that are placed in HTML

pages, and evaluated on the server while the pages are being served. They let you add dynamically generated content to an existing HTML page, without having to serve the entire page via a CGI program, or other dynamic technology.

See: [Server Side Includes \(SSI\)](#)

Per-user web directories

On systems with multiple users, each user can be permitted to have a web site in their home directory using the [UserDir](#) directive. Visitors to a URL `http://example.com/~username/` will get content out of the home directory of the user "username", out of the subdirectory specified by the [UserDir](#) directive.

See: [User web directories \(public html\)](#)

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)