WASD VMS Web Services - Features and Facilities

July 2016

For version 11.0 release of WASD VMS Web Services.

Versions 2.1 (May 1995) through to 9.3 (March 2008) provided the content of this document as the ‘‘Technical Overview’’. Over the years this had inflated to some three hundred pages and in an effort to make it more manageable was distributed between this document, ‘‘Features and Facilities’’, and one containing ‘‘Install and Config’’ details.

Abstract

This document describes the more significant features and facilities available with the WASD Web Services package.

For installation, update and detailed configuration information see ‘‘WASD Web Services - Install and Config’’

For information on CGI, CGIplus, ISAPI, OSU, etc., scripting, see ‘‘WASD Web Services - Scripting’’

And for a description of WASD Web document, SSI and directory listing behaviours and options, ‘‘WASD Web Services - Environment’’

It is strongly suggested those using printed versions of this document also access the Web version. It provides online access to some examples, etc.

Author

Mark G. Daniel

Mark.Daniel@wasd.vsm.com.au

A pox on the houses of all spammers. Make that two poxes.

Online Search

online search

Online PDF

This book is available in PDF for access and subsequent printing by suitable viewers (e.g. Ghostscript) from the location WASD_ROOT:[DOC.FEATURES]WASD_FEATURES.PDF

Online Demonstrations

Some of the online demonstrations may not work due to the local organisation of the Web environment differing from WASD where it was originally written.

WASD VMS Web Services

Copyright © 1996-2016 Mark G. Daniel.

The Apache Group

Bailey Brown Jr.

Clark Cooper, et.al.

```
The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.
```

Bjoern Hoehrmann

This package uses essential algorithm and code from Flexible and Economical UTF-8 Decoder.

```
Copyright (c) 2008-2009 Bjoern Hoehrmann <bjoern@hoehrmann.de>

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.
```

Eric A. Young

This package *can* include cryptographic software written by Eric Young (eay@cryptsoft.com) and Tim Hudson (tjh@cryptsoft.com).

```
This library is free for commercial and non-commercial use provided ...
Eric Young should be given attribution as the author ...
copyright notice is retained
```

Free Software Foundation

This package contains software made available by the Free Software Foundation under the GNU General Public License.

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2, or (at your option)
any later version.
```

Ohio State University

This package contains software provided with the OSU (DECthreads) HTTP server package, authored by David Jones:

```
Copyright 1994,1997 The Ohio State University.
The Ohio State University will not assert copyright with respect
to reproduction, distribution, performance and/or modification
of this program by any person or entity that ensures that all
copies made, controlled or distributed by or for him or it bear
appropriate acknowlegement of the developers of this program.
```

OpenSSL Project

This product *can* include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

```
Redistribution and use in source and binary forms, with or without
modification, are permitted ...
```

Paul E. Jones

This package uses SHA-1 hash code.

```
Copyright (C) 1998, 2009
Paul E. Jones <paulej@packetizer.com>

Freeware Public License (FPL)

This software is licensed as "freeware."  Permission to distribute
this software in source and binary forms, including incorporation
into other products, is hereby granted without a fee.
```

RSA Data Security

This software contains code derived in part from RSA Data Security, Inc:

```
permission granted to make and use derivative works provided that such
works are identified as "derived from the RSA Data Security, Inc.
MD5 Message-Digest Algorithm" in all material mentioning or referencing
the derived work.
```

Stuart Langridge

SortTable version 2
Stuart Langridge, http://www.kryogenix.org/code/browser/sorttable/

```
Thanks to many, many people for contributions and suggestions.
Licenced as X11: http://www.kryogenix.org/code/browser/licence.html
This basically means: do what you want with it.
```

Tatsuhiro Tsujikawa

nghttp2 - HTTP/2 C Library
Tatsuhiro Tsujikawa, https://github.com/tatsuhiro-t

```
Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
```

Other

OpenVMS, **HP TCP/IP Services for OpenVMS**, **HP C**, **Alpha** and **VAX** are registered trademarks of Hewlett Packard Corporation and Hewlett-Packard Development Company, L.P.

MultiNet is a registered trademark of Process Software Corporation.

Pathway is a registered trademark of Attachmate, Inc.

TCPware is a registered trademark of Process Software Corporation.

Contents

Chapter 1 Introduction

Chapter 2 Package Overview

Chapter 3 Authentication and Authorization

Chapter 4 Transport Layer Security

Chapter 5 HTTP/2

Chapter 6 WebDAV

Chapter 7 Proxy Services

Chapter 8 Instances and Environments

Chapter 9 Server Administration

Chapter 10 WATCH Facility

Chapter 11 Server Performance

Chapter 12 HTTPd Web Update

Chapter 13 Utilities and Facilities

Chapter 1
Introduction

With the installation, update and detailed configuration of the WASD Web Services package provided in "WASD Web Services - Install and Config" why have an introduction in this subsequent document? After getting the basics up and running (often the first thing we want to do) it's time to stop and consider the tool and what we're trying to accomplish with it. So this section provides an overview of the package's design philosophy, history and significant features and capabilities by topic.

The document **assumes** a basic understanding of Web technologies and uses terms without explaining them (e.g. HTTP, HTML, URL, CGI, SSI, etc.) The reader is refered to documents specifically on these topics.

It is strongly suggested those using printed versions of this document also access the Web version. It provides online demonstrations of some concepts.

Objectives

WASD Web Services originated from a 1993 decision by Wide Area Surveillance Division (WASD) management (then High Frequency Radar Division, HFRD) to make as much information as possible, both administrative and research, available online to a burgeoning personal desktop workstation and PC environment (to use the current term . . . an *intranet*) using the then emerging Web technologies.

It then became the objective of this author to make *all* of our systems' VMS-related resources available via HTTP and HTML, regardless of the underlying data or storage format. An examination of the WASD package will show that this objective is substantially achieved.

Reasons For Yet Another Web Package

Reasons for developing (remember; back in 1994!) a local HTTP server were few but compelling:

- The WASD Web implementation began mid-1994.
- It was prefered to support this environment on a VMS platform; at the time the most widely used and accessible environment within WASD.

- At that time servers (and even then there were quite a few variations) were largely Unix based, although it was being supported (to a greater or lesses extent) across a wide range of platforms. Ports to VMS, if they existed, were often in-progress or half-baked, employing *Unix*isms that don't translate elegantly to the VMS environment.
- The VMS version of the CERN server (3.0-6) was evaluated during mid-1994:
  - It was (still is) not multi-threaded under VMS (i.e. cannot support concurrent clients). For example, a lengthy search may delay other clients for unacceptable periods.
  - The performance was good with document transfers, but became poor when running a *script*.
  - It is acknowleged in the release notes that it cannot handle a client cancelling a data transfer (a not-uncommon action). This was confirmed experimentally.
- An early version of the OSU server was evaluated via documentation mid-1994. The author considered that the DECthreads of the time to have limitations (including frequent, show-stopping bugs) and OSU had a number of implementation idiosyncracies (e.g. DECnet based scripting).
- HTTP, in the then standard implementation (HTTP/1.0, RFC1945), was relatively simple to implement to the level required to support intra-Divisional requirements.
- Since that time . . .
  - **As of December 1995** the server has worked extremely well and has a number of facilities tailored for the VMS environment. It can continue to be utilized until there are overwhelming reasons for implementing something else.
  - **June 1997** the server and associated software continues to evolve and provide a stable and effective VMS Web environment, even with the advent of a small number of commercial VMS Web products.
  - **October 1999** the package is beginning to mature as an HTTP/1.0 solution, providing not only a fast and stable server but an increasingly extensive collection of applications and tools.
  - **July 2002** it continues to be refined and extended. A greater emphasis on "commercial" functionality has occured over the past couple of years.
  - **December 2004** it now complies with the HTTP/1.1 specification (RFC2616) and provides a very respectable range of functionality and the fastest and most efficient serving environment for VMS.
  - **A decade on (2014)** it continues to be adopted by sites wanting fast, efficient, capable and often philosophically VMS infrastructure. WASD continues to be enhanced and bug-fixed two decades after its initial, tentative steps into the World-Wide information Web.
  - **May 2016** brings HTTP/2 (RFC 7540, RFC 7541) to WASD. A replacement for how HTTP is expressed "on the wire", it is not a ground-up rewrite of the protocol; HTTP methods, status codes and semantics are the same. The focus of the protocol is on performance; specifically, end-user perceived latency, network and server resource usage.

Chapter 2
Package Overview

The most fundamental component of the WASD VMS Web Services environment is the HTTP server (HyperText Transport Protocol Daemon, or HTTPd). WASD has a single-process, multi-threaded, asynchronous I/O design.

The following bullet-points summarise the features and facilities, many of which are described in significant detail in following chapters.

General

- concurrent, multi-threaded client support
- HTTP/2 compliant (RFC 7540, RFC 7541)
- HTTP/1.1 compliant (RFC 2616, RFC 7230 and family)
- HTTP/1.0 compliant (RFC 1954)
- WebDAV 1,2 support (RFC 4918)
- virtual services (servers)
- IPv4 and IPv6 support (requires underlying TCP/IP support)
- requests above a configurable limit can be queued (“throttling”)
- enhanced privacy using Secure Sockets Layer (SSL) technology, including OpenSSL Toolkit, WASD OpenSSL, and HP SSL (Secure Sockets Layer) for OpenVMS Alpha, Itanium and (from late 2003) VAX product
- serves ODS-2 and ODS-5 (EFS) volumes, as well as file names encoded using the PATHWORKS 4/5, Advanced Server (PATHWORKS 6) and SRI (MultiNet NFS, etc.) schemas
- versatile directory listing (generic and VMS-style)
- Server-Side Includes (SSI HTML pre-processing)
- configurable cache, with time-based and forced revalidation (reload)

- byte-range support with 206 partial responses (useful for PDF and restarting file download by modern browsers)
- proxy serving, with local file-system caching, plus the CONNECT method (also allowing a number of esoteric SSL tunnelling configurations), along with FTP proxy
- gatewaying between Web protocols (HTTP-to-SSL, SSL-to-HTTP, HTTP-to-FTP)
- gatewaying between IP protocols (IPv4-to-IPv6, IPv6-to-IPv4)
- clickable-image support (both NCSA and CERN formats)

Scripting

- CGI 1.1 compliant scripting (RFC 3875)
- non-server and user account scripting
- "CGIplus" scripting (offering reduced latency, increased throughput and reduced system impact)
- "Persistent" scripting, Run-Time Environments (RTEs) that provide for simple persistent scripting
- WebSocket scripting environment; a capability introduced with HTML5, providing an asynchronous, bidirectional, full-duplex connection.
- "ISAPI" extensions/scripting (also offering reduced latency, increased throughput and reduced system impact)
- DECnet-based CGI scripting (with connection reuse)
- OSU (DECthreads server) scripting emulation, with connection reuse (as per OSU 3.3a), allowing many OSU scripts to be employed unmodified
- script processor (e.g. PERL, PHP, Python) configurable on file type (suffix)
- configurable, automatic, MIME content-type initiated scripting ("presentation" scripting)

Access Control

- host-level, on per-host or per-domain
- "Basic" and "Digest" user authentication and path/group-based authorization
- WASD-specific user databases
- SYSUAF-authentication and VMS user security profile based file access control
- ACME service authentication (on applicable platforms)
- X.509 client certificate authentication (for SSL transactions)
- RFC 1413 (*ident* daemon) "authentication"
- Example LDAP authenticators

Administration

- multiple *instances* (server processes) executing on the one system allow continuous availability via rolling restarts and “fail-through” processing
- “one-button” control of multiple *instances* on both single systems and across clusters
- online server configuration, including reports on requests, loaded configuration, mapping rules, authorization information and graphical activity displays
- online, live server processing event report (WATCH)
- Web-standard, “common” and “combined” access log formats (allowing processing by most log-analysis tools), along with a user-definition capability allowing custom log formats
- logging periods, where log files automatically change on a daily, weekly or monthly basis (keeps log files ordered and at a managable size)
- customizable message database (capable of supporting non-English and concurrent, multiple languages)

2.1 Server Behaviour

The technical aspects of server design and behaviour are described in “WASD_ROOT:[SRC.HTTPD]READMO

2.2 VMS Versions

The WASD server is supported on any VMS version from V7.0 upwards, on Alpha, Itanium and VAX architectures. The current version (as of late 2014), V8.4 Alpha and Itanium, as is commonly the case on VMS platforms, required nothing more than relinking. Obviously no guarantees can be made for yet-to-be-released versions but at a worst-case these should only require the same.

Up until v10.1 WASD was supported on VMS V6.0 and later. Eventually it had to be dragged kicking and screaming into the mid-1990s!

The WASD distribution and package organisation fully supports mixed-architecture clusters (Alpha, Itanium and/or VAX in the one cluster) as one integrated installation.

2.3 TCP/IP Packages

The WASD server uses the Compaq TCP/IP Services (UCX) BG $QIO interface. The following packages support this interface and may be used.

- TCP/IP Services for OpenVMS (Hewlett Packard Corporation), any version
- Digital TCP/IP Services for OpenVMS (aka UCX), any version
- MultiNet for OpenVMS (Process Software Corporation), any version
- TCPware (Process Software Corporation), any version

To deploy IPv6 services this package must support IPv6.

2.4 International Features

WASD provides a number of features that assist in the support of non-English and multi-language sites. These "international" features only apply to the server, not necessarily to any scripts!

- **Language Variants**

  A directory may contain language-specific variants of a basic document. When requesting the basic document name these variants are automatically and transparently provided as the response if one matches preferences expresses in the request's "Accept-Language:" request header field. Both text and non-text documents (e.g. images) may be provided using this mechanism.

  Configuration information is provided in "WASD Web Services - Install and Config" .

- **Character Sets**

  Generally the default character set for documents on the Web is ISO-8859-1 (Latin-1). The server allows the specification of any character set as a default for text document responses (plain and HTML). In addition, text document file types may be modified or additional ones specified that have a different character set associated with that type. Furthermore, specific character sets may be associated with mapping paths. A site can therefore relatively easily support multiple character set document resources.

  In addition the server may be configured to dynamically convert one character set to another during request processing. This is supported using the VMS standard NCS character set conversion library.

  For further information see [CharsetDefault], [CharsetConvert] and [AddType] in "WASD Web Services - Install and Config" .

- **Server Messages**

  The server uses an administrator-customizable database of messages that can contain multiple language instances of some or all messages, using the Latin-1 character set (ISO8859-1). Although the base English messages can be completely changed and/or translated to provide any message text required or desired, a more convenient approach is to supplement this base set with a language-specific one.

  One language is designated the prefered language. This would most commonly be the language appropriate to the geographical location and/or clientele of the server. Another language is designated the base language. This must have a complete set of messages and is a fall-back for any messages not configured for the additional language. Of course this base language would most commonly be the original English version.

  More than just two languages can be supported. If the browser has *prefered languages* set the server will attempt to match a message with a language in this preference list. If not, then the server-prefered and then the base language message would be issued, in that order. In this way it would be possible to simultaneously provide for English, French, German and Swedish audiences, just for example.

  For message configuration information see "WASD Web Services - Install and Config" .

- **Server Dates**

Dates appearing in server-generated, non-administrative content (e.g. directory listings, not META-tags, which use Web-standard time formats) will use the natural language specified by any SYS$LANGUAGE environment in use on the system or specifically created for the server.

- **Virtual Services**

  Virtual-server-associated mapping, authorization and character-sets allow for easy multiple language and environment sites. Further per-request tailoring may be deployed using conditional rule mapping described below. Single server can support multi-homed (host name) and multiple port services.

  For virtual services information see “WASD Web Services - Install and Config” .

- **Conditional Rule Mapping**

  Mapping rules map requested URL paths to physical or other paths (see “WASD Web Services - Install and Config”). Conditional rules are only applied if the request matches criteria such as prefered language, host address (hence geographical location to a certain extent), etc. This allows requests for generic documents (e.g. home pages) to be mapped to language versions appropriate to the above criteria.

  For conditional mapping information see “WASD Web Services - Install and Config” .

Chapter 3

Authentication and Authorization

Authentication is the verification of a user's identity, usually through username/password credentials. **Authorization** is allowing a certain action to be applied to a particular path based on authentication of the originator.

Generally, authorization is a two step process. First authentication, using a username/password database. Second authorization, determining what the username is allowed to do for this transaction.

Basic authorization was discussed in "WASD Web Services - Install and Config"). This section discusses all the aspects of WASD authentication and authorization.

Overview

By default, the logical name **WASD_CONFIG_AUTH** locates a common authorization rule file. Simple editing of the file and reloading into the running server changes the processing rules.

Server authorization is performed using a configuration file, authentication source, and optional full-access and read-only authorization grouping sources, and is based on per-path directives. There is no user-configured authorization necessary, or possible! In the configuration file paths are associated with the authentication and authorization environments, and so become subject to the HTTPd authorization mechanism. Reiterating . . . WASD HTTPd authorization administration involves those two aspects, setting authorization against paths and administering the authentication and authorization sources.

Authorization is applied to the request path (i.e. the path in the URL used by the client). Sometimes it is possible to access the same resource using different paths. Where this can occur care must be exercised to authorize all possible paths.

Where a request will result in script activation, authorization is performed on both script and path components. First script access is checked for any authorization, then the path component is independently authorized. Either may result in an authorization challenge/failure. This behaviour can be disabled using a path SETting rule, see "WASD Web Services - Install and Config").

The **authentication source** name is refered to as the *realm*, and refers to a collection of usernames and passwords. It can be the system's SYSUAF database.

The **authorization source** is refered to as the *group*, and commonly refers to a collection of usernames and associated *permissions*.

3.1 Rule Interpretation

The configuration file rules are scanned from first towards last, until a matching rule is encountered (or end-of-file). Generally a rule has a trailing wildcard to indicate that all sub-paths are subject to the same authorization requirements.

String Matching

Rule matching is string pattern matching, comparing the request specified path, and optionally other components of the request when using configuration conditionals (see "WASD Web Services - Install and Config"), to a series of patterns, until one of the patterns matches, at which stage the authorization characteristics are applied to the request and authentication processing is undertaken. If a matching pattern (rule) is not found the path is considered not to be subject to authorization. Both wildcard and regular expression based pattern matching is available (see "WASD Web Services - Install and Config").

3.2 Authentication Policy

A *policy* regarding when and how authorization can be used may be established on a per-server basis. This can restrict authentication challenges to "https:" (SSL) requests (Chapter 4), thereby ensuring that the authorization environment is not compromised by use in non-encrypted transactions. Two server qualifiers provide this.

- **/AUTHORIZE=**
  - **ALL** restricts **all** requests to authorized paths. If a path does not have authorization configured against it it is automatically denied access. This is an effective method of preventing inadvertant access to areas in a site (Section 3.14).
  - **SSL** restricts **all** authentication/authorization transactions to the SSL environment.
  - **(SSL,ALL)** combines the above two.
- **/SYSUAF=**
  - Used without any keywords, this qualifier allows all current (non-expired, non-disusered, etc.), non-privileged accounts to be used for authentication purposes.
  - **ID** restricts SYSUAF authenticated account to those possessing a specific VMS resource identifier (Section 3.10.3).
  - **PROXY** allows non-SYSUAF to SYSUAF username proxying (Section 3.10.5).
  - **RELAXED** allows **any** current account to be authorized via the SYSUAF. **This is not recommended,** use rights identifiers to allow some discrimination to be exercised.
  - **SSL** restricts only SYSUAF authenticated transactions to the SSL environment.

- **VMS** allows a combination of all current (non-expired, non-disusered, etc.), non-privileged accounts to be used for authentication purposes (the /SYSUAF without keywords behaviour), with the behaviours provided by the ID keyword.
- **WASD** enables the deprecated, "hard-wired" WASD identifier environment available to this server. See Section 3.10.4.
- **(VMS,ID,SSL)** would allow these multiple keywords to be applied, etc.

Note also that individual paths may be restricted to SSL requests using either the mapping conditional rule configuration or the authorization configuration files. See “WASD Web Services - Install and Config” and Access Restriction Keywords.

In addition, the following configuration parameters have a direct role in an established authorization policy.

- **[AuthFailureLimit] [AuthFailurePeriod] [AuthFailureTimeout]** provide a similar break-in detection and evasion as with VMS. These three directives parallel the functions of SYSGEN parameters LGI_BRK_LIM, LGI_BRK_TMO, LGI_HID_TIM. A single authentication failure marks the particular username in the particular realm as suspect. Repeated failures up to [AuthFailureLimit] attempts within the [AuthFailurePeriod] period puts it into break-in evasion mode after which the period [AuthFailureTimeout] must expire before further attempts have authentication performed and so have any chance to succeed. (This is a change in behaviour to versions earlier than 8.3.) If any of the above three parameters are not specified they default to the corresponding SYSGEN parameter.
- **[AuthRevalidateLoginCookie]** When user revalidation is in effect (see immediately below), after having previously closed the browser initial authentication of a resource is immediately followed by another if a cached entry on the server indicated revalidation was required. This prevents this second request. Requires that browser cookies be enabled.
- **[AuthRevalidateUserMinutes]** sets the number of minutes between successive authentication attempts before the user is forced to reenter the authentication data (via a browser dialog). Zero disables this function. When enabling this feature is is inevitable that [AuthRevalidateLoginCookie] will need to be enabled as well (described immediately above). This is used to suppress an unavoidable second username/password prompt from the browser.

Authentication Cache and Revalidation

User revalidation relies on an entry being maintained in the authentication cache. Each time the entry is flushed, for whatever reason (cache congestion, command-line purge, server restart, etc.), the user will be prompted for credentials. It may be necessary to increase the size of the cache by adjusting [AuthCacheEntriesMax] when this facility is enabled.

Authentication Failures

Details of authentication failures are logged to the server process log.

- **%HTTPD-W-AUTHFAIL** indicates a failure to authenticate (incorrect username/password). The number of failures, the realm name, the user name and the originating host are provided. Isolated instances of this are only of moderate interest. Consecutive instances may indicate a user thrashing about for the correct password, but they usually give up before a dozen attempts.
- **%HTTPD-I-AUTHFAILOK** advises that a previous failure to authenticate has now successfully done so. This is essentially informational.
- **%HTTPD-W-AUTHFAILIM** indicates the number of failures have exceeded the [AuthFailureLimit], after which automatic refusal begins. This message should be of concern and the circumstances investigated, especially if the number of attempts becomes excessive.

Failures may also be directed to the OPCOM facility (see "WASD Web Services - Install and Config").

3.3 Permissions, Path and User

Both paths and usernames have permissions associated with them. A path may be specified as read-only, read and write, write-only (yes, I'm sure someone will want this!), or none (permission to do nothing). A username may be specified as read capable, read and write capable, or only write capable. For each transaction these two are combined to determine the maximum level of access allowed. The allowed action is the logical AND of the path and username permissions.

The permissions may be described using the HTTP method names, or using the more concise abbreviations R, W, and R+W.

HTTP Methods

Path/User	DELETE	GET	HEAD	POST	PROPFIND	PUT	Other WebDAV
READ or R	no	yes	yes	no	yes	no	no
WRITE or W	yes	no	no	yes	no	yes	yes
R+W	yes	yes	yes	yes	yes	yes	yes
NONE	no	no	no	no	no	no	no
DELETE	yes	yes	no	no	no	no	no
GET	no	yes	no	no	no	no	no
HEAD	no	no	yes	no	no	no	no

Path/User	DELETE	GET	HEAD	POST	PROPFIND	PUT	Other Web-DAV
POST	no	no	no	yes	no	no	no
PROPFIND	no	no	no	no	yes	no	no
PUT	no	yes	no	no	no	yes	no
Other WebDAV	no	no	no	no	no	no	yes

3.4 Authorization Configuration File

Requiring a particular path to be authorized in the HTTP transaction is accomplished by applying authorization requirements against that path in a configuration file. This is an activity distinct from setting up and maintaining any authentication/authorization databases required for the environment.

By default, the system-table logical name **WASD_CONFIG_AUTH** locates a common authorization configuration file, unless an individual rule file is specified using a job-table logical name. Simple editing of the file changes the configuration. Comment lines may be included by prefixing them with the hash "#" character, and lines continued by placing the backslash character "\" as the last character on a line.

The [IncludeFile] is a directive common to all WASD configuration, allowing a separate file to be included as a part of the current configuration. See (see "WASD Web Services - Install and Config").

Configuration directives begin either with a "[realm]", "[realm;group]" or "[realm;group-r+w;group-r]" specification, with the forward-slash of a path specification, or with a "[AuthProxy]" or "[AuthProxyFile]" introducing a proxy mapping. Following the path specification are HTTP method keywords controlling group and world permissions to the path, and any **access-restricting** request scheme ("https:") and/or host address(es) and/or username(s).

- **REALM**

  Square brackets are used to enclose a [realm;group;group] specification, introducing a new authentication grouping. Within these brackets is specified the realm name (authentication source), and then optional group (authorization source) names separated by semi-colons. All path specifications following this are authenticated against the specified realm database, and permissions obtained from the group "[realm;group]" database (or authentication database if group not specified), until the next [realm;group;group] specification.

  The following shows the format of an authentication source (realm) only directive.

  ```
  [authentication-source]
  ```

  This one, the format of a directive using both authentication and authorization sources (both realm and group).

  ```
  [authentication-source ; authorization-source]
  ```

The third variation, using an authentication, full-access (read and write) and read-only authorization sources (realm and two grouping).

```
[authentication-source ; full-access-source ; read-only-source]
```

The authentication source may also be given a description. This is the text the browser dialog presents during password prompting. See Realm Description in Section 3.5.

- **PATH**

  Paths are usually specified terminated with an asterisk wildcard. This implies that any directory tree below this is included in the access control. Wildcards may be used to match any portion of the specified path, or not at all. Following the path specification are control keywords representing the HTTP methods or permissions that can be applied against the path, and optional access-restricting list of host address(es) and/or username(s), separated using commas. Access control is against either or both the group and the world. The group access is specified first followed by a semi-colon separated world specification. The following show the format of the path directive, see the examples below to further clarify the format.

  ```
  /root/path/  group-access-list,group-permissions ; \
               world-access-list,world-permissions
  ```

- **PROXY**

  The [AuthProxy] and [AuthProxyFile] directives introduces one or more SYSUAF proxy mappings (Section 3.10.5).

The same path cannot be specified against two different realms for the same virtual service. The reason lies in the HTTP authentication schema, which allows for only one realm in an authentication dialog. How would the server decide which realm to use in the authentication challenge? Of course, different parts of a given tree may have different authorizations, however any tree ending in an asterisk results in the entire sub-tree being controlled by the specified authorization environment, unless a separate specification exists for some inferior portion of the tree.

There is a thirty-one character limit on authentication source names.

Reserved Names

The following realm names are reserved and have special functionality.

- **EXTERNAL -** Any authentication and authorization will be done in some way by an external CGI script. None is attempted by the server. The server does pre-processs the supplied "Authorization:" field however and ensures that any request against a path with this realm supplies authorization credentials before any further request processing (script activation) occurs.
- **NONE -** This refers to any request, is not authenticated in a any way, and just marks the path as having been authorized for access (Section 3.14).
- **OPAQUE -** Allows a script generating its own challenge/response and doing all its own "Authorization:" field processing (a little like EXTERNAL but the server does absolutely nothing).

- **PROMISCUOUS -** This realm is only available while the /PROMISCUOUS qualifier is in use (Chapter 9).
- **RFC1413 -** This IETF document describes an identification protocol that can be used as a form of *authentication* within this realm.
- **TOKEN -** A *token* is a short-lived, cookie delivered, representation of authentication established in another context.
- **WORLD -** This refers to any request and is not authenticated in any way, only the permissions associated with the path are applied to the request. The reserved username "WORLD" becomes the authenticated username.
- **VMS -** Use the server system's SYSUAF database to authenticate the username. For "http:" requests the username/password pairs are transmitted encoded but not encrypted, **this is not recommended**. For "https:" requests, using the implicit security offered by SSL (Chapter 4) the use of SYSUAF authentication is considered viable.

  By default accounts with SYSPRV authorized are always rejected to discourage the use of potentially significant usernames (e.g. SYSTEM). Accounts that are disusered, have passwords that have expired, or that are captive or restricted are also automatically rejected.

  The authentication source may be disguised by giving it a specific description. This will the text the browser dialog presents during password prompting. See Realm Description in Section 3.5.

  See Section 3.10 for further information on these topics.
- **X509 -** Uses X.509 v3 certificates (browser client certificates) to establish identity (authentication) and based on that identity control access to server resources (authorization). This is only available for SSL transactions. See Chapter 4 for further information on SSL, and Section 4.3.12 on X509 realm authorization.

Reserved Username

The following username is reserved.

- **WORLD -** If a path is authorized using the WORLD realm the pseudo-authenticated username becomes "WORLD". Any log will reflect this username and scripts will access a WWW_REMOTE_USER containing this value. Although not forbidden, it is not recommended this string be used as a username in other realms.

Access Restriction Keywords

If a host name, protocol identifier or username is included in the path configuration directive it acts to **further** limit access to matching clients (path and username permissions still apply). If more than one are included a request must match each. If multiple host names and/or usernames are included the client must match at least one of each. Host and username strings may contains the asterisk wildcard, matching one or more consecutive characters. This is most useful when restricting access to all hosts within a given domain, etc. In addition a VMS security profile may be associated with the request.

- **Host Names -** may be specified as either alphabetic (if DNS name resolution is enabled, see [DNSlookup] configuration directive) or literal addresses. When a host restriction occurs there is never an attempt to authenticate any associated username. Hence applying host restrictions very effectively prevents an attack from outside the allowed addresses. The reserved word *#localhost* refers to the host name the server is executing on.
- **Network Mask -** The mask is a dotted-decimal network address, a slash, then a dotted-decimal mask or VLSM (variable-length subnet mask). A network mask operates by bitwise-ANDing the client host address with the mask, bitwise-ANDing the network address supplied with the mask, then comparing the two results for equality.
- **Request Scheme -** (protocol) either "http:" or secured via "https:" (SSL)
- **User Names -** are indicated by a leading tilde, the "~" character (similar or username URL syntax).
- **Profile -** a SYSUAF-authenticated username can have its VMS security profile associated with the request. When applied to a path this profile is used to determine access to the file system. The WASD_CONFIG_AUTH configuration file can have the keyword "profile" added to the restriction list (Section 3.10.8). In a manner-of-speaking this keyword lifts a restriction.

For example

```
/web/secret/* *.three.stooges,~Moe,~Larry,~Curly,read
```

restricts read access to Curly, Larry and Moe accessing from within the three.stooges network, while

```
/web/secret/* https:,*.three.stooges,~Moe,~Larry,~Curly,read
```

applies the further restriction of access via "https:" (SSL) only.

These examples show the use of a network mask to restrict based on the source network of the client. The first, four octets supplied as a mask. The second a VLSM used to specify the length of the network component of the address.

```
/web/secret/* https:,#131.185.250.128/255.255.255.192,~Moe,~Larry,~Curly,read
/web/secret/* https:,#131.185.250.128/26,~Moe,~Larry,~Curly,read
```

These examples both specify a 6 bit subnet. With the above examples the host 131.185.250.250 would be accepted, but 131.185.250.50 would be rejected.

Note that it more efficient to place *protocol* and *host* restrictions at the front of a list.

3.5 Authentication Sources

Authentication credentials may be validated against one of several sources, each with different characteristics.

- **VMS Rights Identifier**

  An identifier is indicated by appending a "=ID" to the name of the realm or group. Also refer to Section 3.10.3.

Whether or not any particular username is allowed to authenticate via the SYSUAF may be controlled by that account holding or not holding a particular rights identifier. Placing "=ID" against realm name implies the username must exist in the SYSUAF and hold the specified identifier name.

```
[PROJECT_A=id]
```

When (and only when) a username has been authenticated via the SYSUAF, rights identifiers associated with that account may be used to control the level-of-access within that realm. This is in addition to any identifier controlling authentication itself.

```
[PROJECT_A=id;PROJECT_A_LIBRARIAN=id;PROJECT_A_USER=id]
```

In this example a username would need to hold the PROJECT_A identifier to be able to authenticate, PROJECT_A_LIBRARIAN to write the path(s) (via POST, PUT) and PROJECT_A_USER to be able to read the path(s).

- **VMS Authentication**

  The server system SYSUAF may be used to authenticate usernames using the VMS account name and password. The realm being VMS may be indicated by using the name "VMS", by appending "=VMS" to another name making it a *VMS synonym*, or by giving it a specific description (Realm Description in Section 3.5). Further information on SYSUAF authentication may be found in Section 3.10. These examples illustrate the general idea.

  ```
  [VMS]
  [LOCAL=vms]
  [ANY_NAME_AT_ALL=vms]
  ```

- **ACME**

  Three Authentication and Credential Management Extension (ACME) agents are currently available (as at VMS V8.3 and WASD v9.3), "VMS" (SYSUAF), "MSV1_0" (Microsoft domain authentication used by Advanced Server) and an LDAP kit. There is also an API that will allow local or third-party agents to be developed. WASD ACME authentication is completely asynchronous and so agents that make network or other relatively latent queries will not add granularity into server processing. By default ACME is used to authenticate requests against the SYSUAF on Alpha and Itanium running VMS V7.3 or later (Section 3.10.1).

  For authorization rules explicitly specifying ACME the Domain Of Interpretation (DOI) becomes the realm name, interposed between the relam description and the ACME authentication source keyword. In this first example the DOI is VMS and so all WASD SYSUAF authentication capabilities are available.

  ```
  ["ACME Coyote"=VMS=ACME;JIN_PROJECT=id]
  /a/path/* r+w,https:
  ```

  In the second example authentication is performed using the same credentials as Advanced Server running on the local system.

  ```
  ["PC Users"=MSV1_0=ACME]
  /a/nuther/path/* r+w,https:
  ```

In this final example the DOI is a third-party agent.

```
["More ACME"=THIRD-PARTY=ACME]
/a/different/path/* r+w,https:
```

- **Simple List**

  A plain-text list may be used to provide usernames for group membership. The format is one username per line, at the start of the line, with optional, white-space delimited text continuing along the line (which could be used as documentation). Blank lines and comment lines are ignored. A line may be continued by ending it with a "\" character. These files may, of course, be created and maintained using any plain text editor. They must exist in the WASD_AUTH: directory, have an extension of ".$HTL", and do not need to be world accessible.

  ```
  # the stooges
  curley     Jerome Horwitz
  larry      Louis Feinberg
  moe        Moses Horwitz
  shemp      Samuel Horwitz
  JoeBesser
  JoeDeRita
  ```

  Simple lists are indicated in the configuration by appending a "=LIST" to the name.

  ```
  [VMS;STOOGES=list]
  ```

  It also possible to use a simple list for authentication purposes. The plain-text password is appended to the username with a trailing equate symbol. Although in general this is not recommended as everything is stored as plain-text it may be suitable as an ad hoc solution in some circumstances. The following example shows the format.

  ```
  # silly example
  fred=dancesalittle  Guess who?
  ginger=rogers       No second prizes!
  ```

- **HTA Database**

  These are binary, fixed 512 byte record files, containing authentication and authorization information. HTA databases may be used for authentication and group membership purposes. The content is much the same, the role differs according to the location in the realm directive. These databases may be administered using the online Server Administration facility (Section 9.5) or the HTAdmin command-line utility (Section 13.8). They are located in the WASD_AUTH: directory and have an extension of ".$HTA".

  (Essentially for historical reasons) HTA databases are the default sources for authorization information. Therefore, using just a name, with no trailing "=*something*", will configure an HTA source. Also, and recommended for clearly showing the intention, appending the "=HTA" qualifier specifies an HTA database. The following example show some of the variations.

  ```
  [VMS;PROJECT_A=hta]
  [DEVELOPERS=hta;PROJECT_A=hta]
  ```

- **X.509 Client Certificate**

Uses X.509 v3 certificates (browser client certificates) to establish identity (authentication) and based on that identity control access to server resources (authorization). This is only available for SSL transactions. See Chapter 4 for further information on SSL, and Section 4.3.12 on X509 realm authorization.

- **RFC1413 Indentification Protocol**

  From RFC1413 (M. St.Johns, 1993) . . .

  The Identification Protocol (a.k.a., "ident", a.k.a., "the Ident Protocol") provides a means to determine the identity of a user of a particular TCP connection. Given a TCP port number pair, it returns a character string which identifies the owner of that connection on the server's system.

  and . . .

  The information returned by this protocol is at most as trustworthy as the host providing it OR the organization operating the host. For example, a PC in an open lab has few if any controls on it to prevent a user from having this protocol return any identifier the user wants. Likewise, if the host has been compromised the information returned may be completely erroneous and misleading.

  The Identification Protocol is not intended as an authorization or access control protocol. At best, it provides some additional auditing information with respect to TCP connections. At worst, it can provide misleading, incorrect, or maliciously incorrect information.

  Nevertheless, RFC1413 may be useful for some purposes in some heterogeneous environments, and so has been made available for *authentication* purposes.

```
[RFC1413]
["Descriptions can be used!"=RFC1413;A_PROJECT=list]
```

  The RFC1413 realm generates no browser username/password dialog. It relies on the system supporting the client to return a reliable identification of the user accessing the HTTP server by looking-up the user of the server connection's peer port.

- **Authorization Agent**

  An authorization agent is a CGI-compliant CGIplus script that is specially activated during the authorization processing. Using CGI environment variables it gets details of the request, makes an assessment based on its own internal authentication/authorization processing, and using the script *callout* mechanism returns the results to the server, which then acting on these, allows or denies access.

  Such agents allow a site to develop local authentication/authorization mechnisms relatively easily, based on CGI principles. A discussion of such a development is not within the scope of this section, see the "WASD Web Services - Scripting" document for information on the use of callouts, and the example and working authorization agents provided in the WASD_ROOT:[SRC.AGENT] directory. The description at the beginning of these programs covers these topics in some detail.

  An authorization agent would be configured using something like the following, where the "AUTHAGENT" is the actual script name doing the authorization. This has the the path "/cgiauth-bin/" prepended to it.

```
["Example Agent"=AUTHAGENT_EXAMPLE=agent]
/some/path/or/other/* r+w
```

It is possible to supply additional, per-path information to an agent. This can be any free-form text (up to a maximum length of 63 characters). This might be a configuration file location, as used in the example CEL authenticator. For example

```
["CEL Authenticator"=AUTHAGENT_CEL=agent]
/some/path/or/other/* r+w,param=WASD_ROOT:[LOCAL]CEL1.LIS
/a/nother/path/* r+w,param=WASD_ROOT:[LOCAL]CEL2.LIS
```

Generally authorization agent scripts use 401/WWW-Authorize: transactions to establish identity and credentials. It is possible for an agent to establish identity outside of this using mechanisms available only to itself. In this case it is necessary suppress the usually automatic generation of username/password dialogs using a realm of *agent+opaque*

```
[AUTHAGENT_PAPI=agent+opaque]
/papi/path/or/other/* r+w
/a/nother/papi/path/* r+w
```

An older mechanism required a leading parameter of "/NO401". It is included here only for reference. The *agent+opaque* realm should now always be used.

```
["Another Authenticator"=AUTHAGENT_ANOTHER=agent]
/some/path/or/other/* r+w,param="/NO401 MORE PARAMETERS CAN BE SUPPLIED"
/a/nother/path/* r+w,param="/NO401 OTHER PARAMETERS CAN BE SUPPLIED"
```

It is necessary to have the following entry in the WASD_CONFIG_MAP configuration file:

```
exec+ /cgiauth-bin/* /cgi-bin/*
```

This allows authentication scripts to be located outside of the general server tree if desired.

- **Token**

  A *token* is a short-lived, cookie delivered, representation of authentication established in another context. Originally devised to allow controlled access to very large datasets without the overhead of SSL in the transmission but with access credentials supplied in the privacy of an SSL connection. The cookie contains NO CREDENTIAL data at all and the authenticator manages an internal database of these so it can determine whether any supplied token is valid and when that token has expired. By default (and commonly) token authorisation occurs in non-SSL space (http:) and the credential authorisation in SSL space (https:).

  Token authorisation is described in Section 3.11).

- **Host Group**

  Instead of a list of usernames contained in a database, a group within a realm (either or both *full-access-source* or *read-only-source*, see Section 3.4) may be specified as a host, group of hosts or network mask. This acts to restrict all requests from clients not matching the IP address specification. Unlike the per-path access restrict list (Access Restriction Keywords) this construct applies to all paths in the realm. It also offers relative efficiencies over restriction lists and lends itself to some environments based on per-host identification (e.g. the RFC1413 realm). Note that IP addresses can be *spoofed* (impersonated) so this form of access control should be deployed with some caution.

```
[RFC1413;131.185.250.*]
/path1/to/be/authorized/* r+w

[RFC1413;131.185.250.0/24]
/path2/to/be/authorized/* r+w

[RFC1413;131.185.250.0/255.255.255.0]
/path3/to/be/authorized/* r+w
```

The examples of realm specifications above all act to restrict read-write access via the RFC1413 realm to hosts within the 131.185.250.*nnn* subnet.

- **External**

  Generally the WASD model is for the server to perform authorisation processing and so the password never becomes visible at the application level. For scripting environments performing their own authentication the server will decode and parse the request "Authorization:" header for paths under the EXTERNAL realm.

  ```
  [EXTERNAL]
  /some/path/or/other/* r+w
  ```

  The various authentication data are then provided in the CGI variables

  AUTH_TYPE
  AUTH_ACCESS
  AUTH_PASSWORD
  AUTH_REALM
  AUTH_REALM_DESCRIPTION
  HTTP_AUTHORIZATION
  REMOTE_USER

- **Opaque**

  If the script is performing its own authentication and authorisation using the raw request header then the server needs to be advised of this by placing the required paths under the OPAQUE realm.

  ```
  [OPAQUE]
  /another/path/* r+w
  ```

  The server will then provide only the "Authorization:" header data in the cgi variable HTTP_AUTHORIZATION from which the username and password may processed.

Multiple Source Types

A realm directive may contain one or more different types of authorization information source, with the following restrictions.

- Rights identifiers may only be used with SYSUAF authenticated requests. The following combinations would therefore not be allowed.

  ```
  [DEVELOPERS;PROJECT_A=id]
  [DEVELOPERS=hta;LIBRARIAN=id;PROJECT_A=list]
  [STOOGES=list;MOE_HOWARD=id]
  ```

- WASD rights identifiers (deprecated) may only be used for group membership when the /AUTHORIZE=WASD server qualifier has been specified at startup, and the username has been authenticated using a WASD identifier. See Section 3.10.4.

Realm Description

It is possible to supply text describing the authentication realm to the browser user that differs from the actual source name. This may be used to disguise the actual source or to provide a more informative description than the source name conveys.

Prefixing the actual realm source name with a double-quote delimited string (of up to 31 characters) and an equate symbol will result in the string being sent to a browser as the realm description during an authentication challenge. Here are some examples.

```
["the local host"=VMS]
["Social Club"=SOCIAL_CLUB_RW=id]
["Finance Staff"=FINANCE=list]
["Just Another Database"=DBACCESS=hta]
```

Note

The *Digest* authentication scheme uses the realm description at both server and browser in the encrypted password challenge and response. When passwords are stored in an HTA file this realm synonym cannot be changed without causing these passwords to be rendered invalid.

3.6 Realm, Full-Access, Read-Only

WASD authorization offers a number of combinations of access control. This is a summary. Please note that when refering to the *level-of-access* a particular username may be allowed (read-only or full, read-write access), that it is always moderated by the level-of-access provided with a path configured within that realm. See Section 3.3.

- **Authentication Only**

  When a path is controlled by a realm that comprises an authentication source only, as in this example

  ```
  [authentication-source]
  ```

  usernames authenticated using that are granted full (read and write) access.

- **Authentication and Group**

  Where a group membership source is provided following the authentication source, as illustrated in this example

  ```
  [authentication-source;group-source]
  ```

  the level-of-access depends on the source of the group membership. If from a *simple-list* of usernames or via a *VMS rights identifier* the username receives full (read and write) access. If from an HTA database the access is dependent on what is set against that user in the database. It can be either full or read-only.

- **Authentication and Two Groups**

When a second group is specified, as in

```
[authentication-source;group-source;group-source]
```

the authentication is interpreted in a fixed fashion. The first group specified contains usernames to be granted full (read and write) access. The second group read-only access. Should a username occur in both groups full access takes precedence.

The second group may be specified as an asterisk wildcard ("*") which is interpreted as *everyone else* (i.e. everyone else gets read-only access).

3.7 Virtual Servers

As described in "WASD Web Services - Install and Config"), virtual service syntax may be used with authorization mapping to selectively apply rules to one specific service. This example provides the essentials of using this syntax. Note that service-specific and service-common rules may be mixed in any order allowing common authorization environments to be shared.

```
# authorization rules example for virtual servers
[[alpha.example.com:443]]
# ALPHA SSL is the only service permitting VMS (SYSUAF) authentication
[LOCAL=vms]
/web/* https:,r+w ; r
/httpd/-/admin/* ~daniel,https:,r+w
[[beta.example.com:80]]
# BETA has its own HTA database
[BETA_USER=hta]
/web/* r+w ; r
[[gamma.example.com:80]]
# GAMMA likewise
[GAMMA_DEVELOPER=id;PROJECT-A=list]
/web/project/a/* r+w ; r
[GAMMA_DEVELOPER=id;PROJECT-B=list]
/web/project/b/* r+w ; r
[[*]]
# allow anyone from the local subnet to upload to here
[WORLD]
/web/unload/* 131.185.200.*,r+w
```

The online Server Administration facility path authorization report (Section 9.4) provides a selector allowing the viewing and checking of rules showing all services or only one particular virtual server, making it simpler to see exactly what any particular service is authorizing against.

3.8 Authorization Configuration Examples

Mixed case is used in the configuration examples (and should be in configuration files) to assist in readability. Rule interpretation however is completely case-insensitive.

1. In the following example the authentication realm is "WASD", a synonym for SYSUAF authentication, and the permissions group "SOCIALCLUB", a simple list of usernames. The directive allows those authenticated from the WASD realm and in the SOCIALCLUB group full access (read and write), and the world read-only.

   ```
   [WASD=vms;SOCIALCLUB=list]
   /web/socialclub/* r+w ; read
   ```

2. This example illustrates restricting access according internet address. Both the group and world restriction is identical, but the group address is being specified numerically, while the world access is being specified alphabetically (just for the purposes of illustration). This access check is done doing simple wildcard comparison, and makes numerical specifications potentially more efficient because they are usually shorter. The second line restricts that path's write access even further, to one username, "BLOGGS".

```
[WASD=vms;SOCIALCLUB=list]
/web/socialclub/* 131.185.45.*,get,post; *.example.com,get
/web/socialclub/accounts/* 131.185.45.*,~BLOGGS,get,post; *.example.com,get
```

3. Three sources for authorization are specified in the following example. As the authentication source is VMS (by rights identifier), the full-access group and read-only group can also be determined by possessing the specified identifiers. The first path can only be written to by those holding the full-access identifier (librarian), the second path can only be read by both. The world has no access to these paths.

```
[DEVELOPER=id;PROJECT_A_LIBRARIAN=id;PROJECT_A_USER=id]
/web/projects/a/*  r+w
/web/projects/*    r
```

4. This example is the same as the one above, except in this case everyone else (that can authenticate against the resource) gets read-only access to the projects.

```
[DEVELOPER=id;PROJECT_A_LIBRARIAN=id;*]
/web/projects/a/*  r+w
/web/projects/*    r
```

5. In the following example the authentication realm and group are a single HTA database, "ADMIN". The first directive allows those in the ADMIN group to read and write, and the world to read ("get,post;get"). The second line restricts write and even read access to ADMIN group, no world access at all ("get,post").

```
[ADMIN=hta]
/web/everyone/* get,post;get
/web/select/few/* get,post
```

6. With this example usernames are used to control access to the specified paths. These usernames are authenticated from the COMPANY database. The world has read access in both cases. Note the realm description, "The Company".

```
["The Company"=COMPANY=hta]
/web/docs/* ~Howard,~George,~Fred,r+w ; r
/web/accounts/* ~George,r+w ; r
```

7. The following example shows a path specifying the local system's SYSUAF being used to authenticate any usernames. Whenever using SYSUAF authentication it is **strongly recommended to limit the potential hosts** that can authenticate in this way by always using a host-limiting access restriction list. The world gets read access.

```
[VMS]
/web/local/area/* 131.185.250.*,r+w ; r
```

8. To restrict server administration to browsers executing on the server system itself and the SYSUAF-authenticated username DANIEL use a restriction list similar to the following. It also shows the use of SYSUAF-authentication being hidden by using a realm description.

```
["not the VMS SYSUAF"=VMS]
/httpd/-/admin/*  #localhost,~daniel,r+w
```

9. This example uses the RFC1413 *identification protocol* as the authentication source and a host group to control full access to paths in the realm.

```
["Ident Protocol"=RFC1413;131.185.250.0/24]
/web/local/* r+w
```

10. The following example illustrates providing a read and writable area (GET, POST and PUTable) to hosts in the local network **without username authentication** (careful!).

```
[WORLD]
/web/scratch/*  *.local.hosts.only,r+w
```

3.8.1 KISS

WASD authorization allows for very simple authorization environments and provides the scope for quite complex ones. The path authentication scheme allows for multiple, individually-maintained authentication and authorization databases that can then be administered by autonomous managers, applying to widely diverse paths, all under the ultimate control of the overall Web administrator.

Fortunately great complexity is not generally necessary.

Most sites would be expected to require only an elementary setup allowing a few selected Web information managers the ability to write to selected paths. This can best be provided with the one authentication database containing read and write permissions against each user, with and access-restriction list against individual paths.

For example. Consider a site with three departments, each of which wishes to have three representatives capable of administering the departmental Web information. Authentication is via the SYSUAF. Web administrators hold an approriate VMS rights identifier, “WEBADMIN”. Department groupings are provided by three simple lists of names, including the Web administrators (whose rights identifier would not be applied if access control is via a simple list), a fourth lists those with read-only access into the Finance area. The four grouping files would look like:

```
# Department 1            # Department 2
WEB1                      WEB1
WEB2                      WEB2
JOHN                      RINGO
PAUL                      CURLY
GEORGE                    LARRY

# Department 3            # Finance (read access)
WEB1                      PAUL
WEB2                      GEORGE
MOE                       JOHN
SHEMP                     RINGO
MAC
```

The authorization configuration file then contains:

```
##############################################################################

# allow web masters (!) to use the server administration facility
#                       to revise web configuration files
# world has no access (read or write)
# access is only allowed from a browser in the same subnet as the HTTPd
["Hypo Thetical Corp."=HYPOTHETICAL=vms;WEBADMIN=id]
/httpd/-/admin/*  #150.15.30.*,r+w
/wasd_root/local/*  #150.15.30.*,r+w

# allows Department 1 representatives to maintain their web
# this may only be done from within the company subnet
# world has read access
["Hypo Thetical Corp."=HYPOTHETICAL=vms;DEPARTMENT1=list]
/web/dept/general/*   150.15.30.*,r+w ; r

# and so on for the rest of the departments

["Hypo Thetical Corp."=HYPOTHETICAL=vms;DEPARTMENT2=list;FINANCE=list]
# no world read access into finance, only those in the FINANCE list
/web/dept/finance/*    150.15.30.*,r+w

["Hypo Thetical Corp."=HYPOTHETICAL=vms;DEPARTMENT3=list]
/web/dept/inventory/*       150.15.30.*,r+w ; r
/web/dept/production/*      150.15.30.*,r+w ; r
# (the next uses line continuation just for illustration)
/web/dept/marketing/*       150.15.30.*,\
                            r+w ;\
                            read

# we need an area for general POSTing (just for illustration :-)
[WORLD]
/web/world/*  r+w

##############################################################################
```

3.9 Authorization Cache

Access to authentication sources, SYSUAF, simple lists and HTA databases, are relatively expensive operations. To reduce the impact of this activity on request latency and general server performance, authentication and realm-associated permissions for each authenticated username are stored in a cache. This means that only the initial request needs to be checked from appropriate databases, subsequent ones are resolved more quickly and efficiently from cache.

Such cached entries have a finite lifetime associated with them. This ensures that authorization information associated with that user is regularly refreshed. This period, in minutes, is set using the [AuthCacheMinutes] configuration parameter. Zero disables caching with a consequent impact on performance.

Implication

Where-ever a cache is employed there arises the problem of keeping the contents current. The simple lifetime on entries in the authentication cache means they will only be checked for currency whenever it expires. Changes may have occured to the databases in the meantime.

Generally there is are other considerations when adding user access. Previously the user attempt failed (and was evaluated each time), now the user is allowed access and the result is cached.

When removing or modifying access for a user the cached contents must be taken into account. The user will continue to experience the previous level of access until the cache lifetime expires on the entry. When making such changes it is recommended to explicitly purge the authentication cache either from the command line using /DO=AUTH=PURGE (Section 9.7) or via the Server Administration facility (Chapter 9). Of course the other solution is just to disable caching, which is a less than optimal solution.

3.10 SYSUAF-Authenticated Users

The ability to authenticate using the system's SYSUAF is controlled by the server /SYSUAF[=keyword] qualifier. By default it is disabled.

WARNING!

SYSUAF authentication is not recommended except in the most secure of LAN environments or when SSL is employed.

HTTP credentials (username and password) are transmitted as encoded plain-text making them vulnerable to evesdropping.

By default accounts with SYSPRV authorized are always rejected to discourage the use of potentially significant usernames (e.g. SYSTEM). This behaviour can be changed through the use of specific identifiers, see Section 3.10.3 immediately below. Accounts that are disusered, have passwords that have expired or that are captive or restricted are always rejected. Accounts that have access day/time restricting access will have those restrictions honoured (see Section 3.10.3 for a workaround for this).

Also see Section 3.10.6.

3.10.1 ACME

By default the Authentication and Credential Management Extension (ACME) is used to authenticate SYSUAF requests on Alpha and Itanium running VMS V7.3 or later (Section 3.5). VAX and earlier versions of VMS use WASD's own SYSUAF authentication routines. The advantage of ACME is with the processing of the (rather complex) authentication requirements by a vendor-supplied implementation. It also allows SYSUAF password change to be made subject to the full site policy (password history, dictionary checking, etc.) which WASD does not implement.

3.10.2 Logon Type

By default SYSUAF authentication uses the NETWORK access restriction from the account SYSUAF record. Alternatives LOCAL, DIALUP and REMOTE may be specified using global configuration directive

```
# WASD_CONFIG_GLOBAL
[AuthSYSUAFlogonType]  REMOTE
```

and/or authorization rule parameter 'param="logon=REMOTE"'

```
["VMS Credentials"=WASD_VMS_RW=ID]
/secured/* r+w,https,param="logon=REMOTE"
```

(which takes precedence).

3.10.3 Rights Identifiers

Whether or not any particular username is allowed to authenticate via the SYSUAF may be controlled by that account holding or not holding a particular VMS rights identifier. When a username has been authenticated via the SYSUAF, rights identifiers associated with that account may be used to control the level-of-access within that realm.

Use of identifiers for these purposes are enabled using the /SYSUAF=ID server startup qualifier.

The first three reserved identifier names are optional. A warning will be reported during startup if these are not found. The fourth must exist if SYSUAF proxy mappings are used in a /SYSUAF=ID environment.

- **WASD_HTTPS_ONLY -** restricts accounts holding it to authenticating using SSL (https:). Authentication via a standard "http:" will always be denied.
- **WASD_NIL_ACCESS -** allows accounts with access time restrictions to authenticate via the SYSUAF. This is particularly intended to support the use of nil-access accounts, see Section 3.10.6.
- **WASD_PASSWORD_CHANGE -** allows an account to modify its SYSUAF password, if this is configured for the server, see Section 3.15.
- **WASD_PROXY_ACCESS -** allows an account to be used for proxy access if /SYSUAF=ID is in effect, see Section 3.10.5.

Identifiers may be managed using the following commands. If unsure of the security implications of this action consult the relevant VMS system management security documentation.

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD /IDENTIFIER WASD_HTTPS_ONLY
UAF> ADD /IDENTIFIER PROJECT_USER
UAF> ADD /IDENTIFIER PROJECT_DEVELOPER
UAF> ADD /IDENTIFIER PROJECT_LIBRARIAN
```

They can then be provided to desired accounts using commands similar to the following:

```
UAF> GRANT /IDENTIFIER PROJECT_USER <account>
```

and removed using:

```
UAF> REVOKE /IDENTIFIER PROJECT_USER <account>
```

Be aware that, as with all successful authentications, and due to the WASD internal authentication cache, changing database contents does not immediately affect access. Any change in the RIGHTSLIST won't be reflected until the cache entry expires or it is explicitly flushed (Section 3.9).

3.10.4 WASD "Hard-Wired" Identifiers

Deprecated and Discouraged

As this has been deprecated for some years now the documentation for this functionality has been removed.

For backward-reference see the "WASD Hypertext Services - Technical Overview" document for release v9.3 or earlier.

3.10.5 VMS Account Proxying

Any authentication realm can have its usernames mapped into VMS usernames and the VMS username used as if it had been authenticated from the SYSUAF. This is a form of proxy access.

CAUTION

This is an extremely powerful mechanism and as a consequence requires enabling on the command-line at server startup using the /SYSUAF=PROXY qualifier and keyword. If identifiers are used to control SYSUAF authentication (i.e. /SYSUAF=ID) then any account mapped by proxy access must hold the WASD_PROXY_ACCESS identifier described in Section 3.10.3 (and server startup would be something like "/SYSUAF=(ID,PROXY)").

When a proxy mapping occurs request user authorization detail reflects the SYSUAF username characteristics, not the actual original authentication source. This includes username, user details (i.e. becomes that derived from the *owner* field in the SYSUAF), constraints on the username access (e.g. SSL only), and user capabilities including any profile if enabled. Authorization source detail remains unchanged, reflecting the realm, realm description and group of the original source. For CGI scripting an additional variable, WWW_AUTH_REMOTE_USER, provides the original remote username.

For each realm, and even for each path, a different collection of mappings can be applied. Proxy entries are strings containing no white space. There are three basic variations, each with an optional host or network mask component.

1. remote[@host | @network/mask]=SYSUAF
2. *[@host | @network/mask]=SYSUAF
3. *[@host | @network/mask]=*

The *SYSUAF* is the VMS username being mapped to. The *remote* is the remote username (CGI variable WWW_REMOTE_USER). The first variation maps a matching remote username (and optional host/network) onto the specific SYSUAF username. The second maps all remote usernames (and optional host/network) to the one SYSUAF username (useful as a final mapping). The third maps all remote usernames (optionally on the remote host/network) into the same SYSUAF username (again useful as a final mapping if there is a one-to-one equivalence between the systems).

Proxy mappings are processed sequentially from first to last until a matching rule is encountered. If none is found authorization is denied. Match-all and default mappings can be specified.

```
[RFC1413]
[AuthProxy] bloggs@131.185.250.1=fred
[AuthProxy] doe@131.185.250.*=john system=- *@131.185.252.0/24=*
[AuthProxy] *=GUEST
```

In this example the username *bloggs* on system 131.185.250.1 can access as if the request had been authenticated via the SYSUAF using the username and password of *FRED*, although of course no SYSUAF username or password needs to be supplied. The same applies to the second mapping, *doe* on the remote system to *JOHN* on the VMS system. The third mapping disallows a *system* account ever being mapped to the VMS equivalent. The fourth, wildcard mapping, maps all accounts on all systems in 131.185.250.0 8 bit subnet to the same VMS username on the server system. The fifth mapping provides a default username for all other remote usernames (and used like this would terminate further mapping).

Note that multiple, space-separated proxy entries may be placed on a single line. In this case they are processed from left to right and first to last.

```
["Just an Example"=EXAMPLE=list]
[AuthProxy] bloggs@131.185.250.1=fred doe@131.185.250.1=doe system=- \
*@131.185.252.0/24=* *=GUEST
```

Proxy mapping rules should be placed after a realm specification and before any authorization path rules in that realm. In this way the mappings will apply to all rules in that realm. It is possible to change the mappings between rules. Just insert the new mappings before the (first) rule they apply to. This cancels any previous mappings and starts a new set. This is an example.

```
["A Bunch of Users"=USERS=hta]
[AuthProxy] bloggs@131.185.250.1=fred doe@131.185.250.1=john
/fred/and/johns/path/* r+w
[AuthProxy] *=GUEST
/other/path/* read
```

An alternative to in-line proxy mapping is to provide the mappings in one or more independent files. In-line and in-file mappings may be combined.

```
["Another Bunch of Users"=MORE_USERS=hta]
[AuthProxy] SYSTEM=-
[AuthProxyFile]  WASD_ROOT:[LOCAL]PROXY.CONF
/path/for/proxy* r+w
```

To cancel all mappings for following rules use an [AuthProxy] (with no following mapping detail). Previous mappings are always cancelled with the start of a new realm specification. Where proxy mapping is not enabled at the command line or a proxy file cannot be loaded at startup a proxy entry is inserted preventing **all access** to the path.

REMEMBER - proxy processing can be observed using the WATCH facility.

3.10.6 Nil-Access VMS Accounts

It is possible, and may be quite effective for some environments, to have a SYSUAF account or accounts strictly for HTTP authorization, with no actual interactive or other access allowed to the VMS system itself. This would relax the caution on the use of SYSUAF authentication outside of SSL transactions. An obvious use would be for the HTTP server administrator. Additional accounts could be provided for other authorization requirements, all without compromising the system's security.

In setting up such an environment it is vital to ensure the HTTPd server is started using the /SYSUAF=ID qualifier (Section 3.2). This will require all SYSUAF-authenticated accounts to possess a specific VMS resource identifier, accounts that do not possess the identifier cannot be used for HTTP authentication. In addition the identifier WASD_NIL_ACCESS will need to be held (Section 3.10.3), allowing the account to authenticate despite being restricted by REMOTE and NETWORK time restrictions.

To provide such an account select a group number that is currently unused for any other purpose. Create the desired account using whatever local utility is used then activate VMS AUTHORIZE and effectively disable access to that account from all sources and grant the appropriate access identifier (see Section 3.10.3 above).

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> MODIFY <account> /NOINTERACTIVE /NONETWORK /NOBATCH /FLAG=DISMAIL
UAF> GRANT /IDENTIFIER WASD_VMS_RW <account>
```

3.10.7 SYSUAF and SSL

When SSL is in use (Chapter 4) the username/password authentication information is inherently secured via the encrypted communications of SSL. To enforce access to be via SSL add the following to the WASD_CONFIG_MAP configuration file:

```
/whatever/path/you/like/*  "403 Access denied."  ![sc:https]
```

or alternatively the following to the WASD_CONFIG_AUTH configuration file:

```
[REALM]
/whatever/path/you/like/*  https:
```

Note that this mechanism is applied **after** any path and method assessment made by the server's authentication schema.

The qualifier /SYSUAF=SSL provides a powerful mechanism for protecting SYSUAF authentication, restricting SYSUAF authenticated transactions to the SSL environment. The combination /SYSUAF=(SSL,ID) is particularly effective.

Also see Section 3.2.

3.10.8 SYSUAF Security Profile

It is possible to control access to files and directories based on the VMS security profile of a SYSUAF-authenticated remote user. This functionality is implemented using VMS security system services involving SYSUAF and RIGHTSLIST information. The feature must be explicitly allowed using the server /PROFILE qualifier. By default it is disabled.

Note

Use caution when deploying the /PROFILE qualifier. It was really designed with a very specific environment in mind, that of an Intranet where the sole purpose was to provide VMS users access to their normal VMS resources via a Web interface.

When a SYSUAF-authenticated user (i.e. the VMS realm) is first authenticated a VMS security-profile is created and stored in the authentication cache (Section 3.9). A cached profile is an efficient method of implementing this as it obviously removes the need of creating a user profile each time a resource is assessed. If this profile exists in the cache it is attached to each request authenticated for that user. As it is cached for a period, any change to a user's security profile in the SYSUAF or RIGHTSLIST won't be reflected in the cached profile until the cache entry expires or it is explicitly flushed (Section 9.6).

When a request has this security profile all accesses to files and directories are assessed against it. When a file or directory access is requested the security-profile is employed by a VMS security system service to assess the access. If allowed, it is provided via the SYSTEM file protection field. Hence it is possible to be eligible for access via the OWNER field but not actually be able to access it because of SYSTEM field protections! If not allowed, a "no privilege" error is generated.

Once enabled using /PROFILE it can be applied to all SYSUAF authenticated paths, but must be enabled on a per-path basis, using the WASD_CONFIG_AUTH *profile* keyword (Access Restriction Keywords)

```
# WASD_CONFIG_AUTH
[VMS;VMS]
/wasd_root/local/* profile,https:,r+w
```

or the WASD_CONFIG_MAP SET *profile* and *noprofile* mapping rules (see "WASD Web Services - Install and Config").

```
# WASD_CONFIG_MAP
set /wasd_root/local/* profile
set * noprofile
```

Of course, this functionality only provides access for the server, IT DOES NOT PROPAGATE TO ANY SCRIPT ACCESS. If scripts must have a similar ability they should implement their own scheme (which is not too difficult,) see WASD_ROOT:[SRC.MISC]CHKACC.C based on the CGI variable WWW_AUTH_REALM which would be "VMS" indicating SYSUAF-authentication, and the authenticated name in WWW_REMOTE_USER.

Performance Impact

If the /PROFILE qualifier has enabled SYSUAF-authenticated security profiles, whenever a file or directory is assessed for access an explicit VMS security system service call is made. This call builds a security profile of the object being assessed, compares the cached user security profile and returns an indication whether access is permitted or forbidden. This is addition to any such assessments made by the file system as it is accessed.

This extra security assessment is not done for non-SYSUAF-authenticated accesses within the same server.

For file access this extra overhead is negligible but becomes more significant with directory listings ("Index of") where each file in the directory is independently assessed for access.

3.10.9 SYSUAF Profile For Full Site Access

Much of a site's package directory tree is inaccessible to the server account. One use of the SYSUAF profile functionality is to allow authenticated accesss to all files in that tree. This can accomplished by creating a specific mapping for this purpose, subjecting that to SYSUAF authentication with /PROFILE behaviour enabled (Section 3.10.8), and limiting the access to a SYSTEM group account. As all files in the WASD package are owned by SYSTEM the security profile used allows access to all files.

The following example shows a path with a leading dollar (to differentiate it from general access) being mapped into the package tree. The "set * noprofile" limits the application of this to the /$WASD_ROOT/ path (with the inline "profile").

```
# WASD_CONFIG_MAP
set * noprofile
  .
  .
  .
pass /wasd_root/* /wasd_root/*
pass /$WASD_ROOT/* /wasd_root/* profile
```

This path is then subjected to SYSUAF authentication with access limited to an SSL request from a specific IP address (the site administrator's) and the SYSTEM account.

```
# WASD_CONFIG_AUTH
[["/$WASD_ROOT/ Access"=WASD_TREE_ACCESS=id]]
/$WASD_ROOT/* https,10.1.1.2,~system,read
```

3.11 Token Authentication

This is a niche authorisation environment for addressing niche requirements.

A *token* is an HTTP cookie delivered representation of authentication established in another context. Originally devised to allow controlled access to very large datasets without the overhead of SSL in the transmission but with access credentials supplied in the privacy of an SSL connection.

A common scenario is where the client starts off attempting to access a resource in non-SSL space which is controlled by token authentication. In the first instance the authenticator detects there is no access token present and redirects the client (browser) to the SSL equivalent of that space, where credentials can be supplied encrypted. In this example

scenario the SSL area is controlled by WASD SYSUAF authentication (can be SSL client certificate, etc.) and the username/password is prompted for. When correctly entered this generates a token. The token is stored (with corresponding detail) as a record in a server-internal database and then returned to the browser as a set-cookie value.

With the token data stored the browser is transparently redirected back to the non-SSL space where the actual access is to be undertaken, this time the browser presenting the cookie containing the token. The authenticator examines the token, looking it up in the database. If found, has originated from the same IP address, represents the same authentication realm, and has not expired, it then allows the non-SSL space access to proceed, and in this example scenario the dataset transfer is initiated (in unencrypted clear-text). If the token is not found in the database or has expired, then the process is repeated with a redirect back into SSL space. If the realms differ a 403 forbidden response is issued (see configuration below).

The token itself is a significant sequence of pseudo-random characters, is short-lived (configurable as anything from a few seconds to a few tens of seconds, or more), and as a consequence is frequently regenerated. The token is just that, containing no actual credential data at all. It might be possible to snoop but as it contains nothing of value in itself, expires relatively quickly, and has an originating IP address check, the fairly remote risk of playback is just that.

The authenticator does all the work, implicitly redirecting the user from non-SSL space to SSL space for the original authentication, and then back again with the token used for access in the non-SSL space. With the expiry of a token it undertakes that cycle again, redirecting back to the SSL-space where the browser-cached credentials will be supplied automatically allowing the fresh token to be issued, and then redirected back into non-SSL space for access. To emphasise - all this is transparent to the user.

As a consequence of this model the resource being controlled can ONLY be accessed from non-SSL space using the controlled path. To access the same resource from SSL space a distinct path to the resource must be provided.

Configuration

As token authorisation relies on the client agent having HTTP cookies enabled (globally or specifically for the site) it is useful to have this tested for and/or advised about, on some related but other area of the site. There are simple techniques using JavaScript for detecting the availability of cookie processing. Search the Web for a suitable solution.

The automatic authorisation and redirection occurs using a combination of two distinguishable authorisation rules, one for supplying the credentials, the other for using the token for authorisation. In this example (and commonly) the resources are at "/location/" and the configuration accepts user-supplied credentials in SSL space and uses the token in non-SSL space. The asterisk just indicates that in the absence of any other parameter this authorisation rule has a complementary token rule where access will actually occur.

```
# WASD_CONFIG_AUTH
if (ssl:)
   ["VMS credentials"=WASD_VMS_RW=id+"TOKEN=*"]
   /location/* r+w
else
   [WASD_VMS_RW=TOKEN]
   /location/* r+w
endif
```

And in this example, the same arrangement but with non-standard ports (specified using an integer with a leading colon).

```
# WASD_CONFIG_AUTH
if (ssl:)
   ["VMS credentials"=WASD_VMS_RW=id+"TOKEN=:7080"]
   /location/* r+w
else
   [WASD_VMS_RW=TOKEN+"TOKEN=:7443"]
   /location/* r+w
endif
```

To prevent potential thrashing, where multiple, distinct realms within a *single* request are authorised using tokens, corresponding multiple token (cookie) names must be used. It is expected that this would be an uncommon but not impossible scenario. The “thrashing” would be a result of authorisation associated with a single, particular token name. Where a realm differs from a previous token generated another is required. The token authorisation scheme forces the use of distinct token names by 403-forbidding change of realm using the one token. Use explicitly specified, independent token (cookie) names, or an integer preceded by an ampersand (which appends the integer to the base token name), ensuring the complementary rules are using the same name/integer.

```
# WASD_CONFIG_AUTH
if (ssl:)
   ["VMS credentials"=WASD_VMS_RW=id+"TOKEN=&42"]
   /location/* r+w
else
   [WASD_VMS_RW=TOKEN+"TOKEN=&42"]
   /location/* r+w
endif
```

For the final example, the token is contained in the non-default cookie named "WaSd_example" and the authentication performed using an X509 client certificate (which can only be supplied via SSL).

```
# WASD_CONFIG_AUTH
if (ssl:)
   [X509+"TOKEN=WaSd_example"]
   /location/* r+w
else
   [X509=TOKEN+"TOKEN=WaSd_example"]
   /location/* r+w
endif
```

Some additional detail is available from the AUTHTOKEN.C code module.

3.12 Skeleton-Key Authentication

Provides a username and password that is authenticated from data placed into the global common (i.e. in memory) by the site administrator. The username and password expire (become non-effective) after a period, one hour by default or an interval specified when the username and password are registered.

It is a method for allowing ad hoc authenticated access to the server, primarily intended for non-configured access to the online Server Administration facilities (Section 9.1) but is available for other purposes where a permanent username and password in an authentication database is not necessary. A skeleton-key authenticated request is subject to all other authorization processing (i.e. access restrictions, etc.), and can be controlled using the likes of '~_*', etc.

The site administrator uses the command line directive

```
$ HTTPD /DO=AUTH=SKELKEY=_username:password[:period]
```

to set the username/password, and optionally the period in minutes. This authentication credential can be cancelled at any time using

```
$ HTTPD /DO=AUTH=SKELKEY=0
```

The username must begin with an underscore (to reduce the chances of clashing with a legitimate username) and have a minimum of 6 other characters. The password is delimited by a colon and must be at least 8 characters. The optional period in minutes can be from 1 to 10080 (one week). If not supplied it defaults to 60 (one hour). After the period expires the skeleton key is no longer accepted until reset.

Note

Choose username and password strings that are less-than-obvious and a period that's sufficient to the task! After all, it's **your site** that you might compromise!

The authentication process (with skeleton-key) is performed using these basic steps.

1. Is a skeleton-key set? If not continue on with the normal authentication process.
2. If set then check the request username leading character for an underscore. If not then continue on with normal authentication.
3. If it begins with an underscore then match the request and skeleton-key usernames. If they do not match then continue with normal authentication.
4. If the usernames match then compare the request and skeleton-key passwords. If matched then it's authenticated. If not it becomes an authentication failure.

Note that the authenticator resumes looking for a username from a configured authentication source unless the request and skeleton-key usernames match. After that the passwords either match allowing access or do not match resulting in an authentication failure.

Examples

```
$ HTTPD /DO=AUTH=SKELKEY=_FRED2ACC:USE82PA55
$ HTTPD /DO=AUTH=SKELKEY=_ANDY2WERP:EGGO4TEE:10
```

3.13 Controlling Server Write Access

The server account should have no direct write access to into any directory structure. Files in these areas should be owned by SYSTEM ([1,4]). Write access for the server into VMS directories (using the POST or PUT HTTP methods) should be controlled using VMS ACLs. **This is in addition to the path authorization of the server itself of course!** The recommendation to have no ownership of files and provide an ACE on required directories prevents inadvertant mapping/authorization of a path resulting in the ability to write somewhere not intended.

Two different ACEs implement two grades of access.

1. If the ACE grants **CONTROL** access to the server account then only VMS-authenticated usernames with security profiles can potentially write to the directory. Only potentially, because a further check is made to assess whether that VMS account in particular has write access.

   This example shows a suitable ACE that applies only to the original directory:

   ```
   $ SET SECURITY directory.DIR -
     /ACL=(IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
   ```

   This example shows setting an ACE that will propagate to created files and importantly, subdirectories:

   ```
   $ SET SECURITY directory.DIR -
     /ACL=((IDENT=HTTP$SERVER,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL), -
           (IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL))
   ```

2. If the ACE grants **WRITE** access then the directory can be written into by any authenticated username for the authorized path.

   This example shows a suitable ACE that applies only to the original directory:

   ```
   $ SET SECURITY directory.DIR -
     /ACL=(IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE)
   ```

   This example shows setting an ACE that will propagate to created files and importantly, subdirectories:

   ```
   $ SET SECURITY directory.DIR -
     /ACL=((IDENT=HTTP$SERVER,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE+DELETE), -
           (IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE))
   ```

To assist with the setting of the required ACEs an example, general-purpose DCL procedure is provided, WASD_ROOT:[EXAMPLE]AUTHACE.COM.

3.14 Securing All Requests

Some sites may be sensitive enough about Web resources that the possibility of providing inadvertant access to some area or another is of major concern. WASD provides a facility that will automatically deny access to any path that does not appear in the authorization configuration file. This does mean that all paths requiring access must have authorization rules associated with them, but if something is missed some resource does not unexpectedly become visible.

At server startup the /AUTHORIZE=ALL qualifier enables this facility.

For paths that require authentication and authorization the standard realms and rules apply. To indicate that a particular path should be allowed access, but that no authorization applies the “NONE” realm may be used. The following example provides some indication of how it should be used.

```
# allow the librarian to update this area, world to read it
[VMS;LIBRARIAN=id]
/web/library/* r+w ; read
# indicate there is no authorization to be applied
[NONE]
# allow access to general web areas
/web/*
# allow access to the WASD_ROOT tree
/wasd_root/*
```

There is also a per-path equivalent of the /AUTHORIZE=ALL functionality, described in “WASD Web Services - Install and Config” . This allows a path tree to be require authorization be enabled against it.

```
# avoid an absence of authorization allowing unintentional access
set /web/sensitive/* auth=all
```

3.15 User Password Modification

The server provides for users to be able to change their own HTA passwords (and SYSUAF if required). This functionality, though desirable from the administrator’s viewpoint, is not mandatory if the administrator is content to field any password changes, forgotten passwords, etc. Keep in mind that passwords, though not visible during entry, are passed to the server using clear-text form fields (which is why SSL is recommended).

Password modification is enabled by including a mapping rule to the internal change script. For example:

```
pass /httpd/-/change/* /httpd/-/change/*
```

Any database to be enabled for password modification must have a writable authorization path associated with it. For example:

```
[GROUP=id;GROUP=id]
/httpd/-/change/group/* r+w

[ANOTHER_GROUP=id;ANOTHER_GROUP=id]
/httpd/-/change/another_group/* r+w
```

Note

What looks like redundancy in specifying an identical realm and group authorization is what allows multiple, independant identifiers to be individually controlled for password change (i.e. one group of identifier holders allowed to change the password, another not).

Use some form of cautionary wrapper if providing this functionality over something other than an Intranet or SSL connection:

```
<H2>Change Your Authentication</H2>

<blockquote>
Change the password used to identify yourself to the REALM Web environment for
some actions.  Note that this <U>not</U> an operating system password, nor has
it anything to do with it.  Due to the inherent weaknesses of using
non-encrypted password transmissions on networks <FONT COLOR="#ff0000"><U>DO
NOT</U> use a password you have in use anywhere else, especially an operating
system password!</FONT> You need your current password to make the change.  If
you have forgotten what it is contact <A HREF="/web/webadmin.html">WebAdmin</A>,
preferably via e-mail, for the change to be made on your behalf.
</blockquote>

<UL>
<LI><A HREF="/httpd/-/change/REALM/">REALM</A> realm.
</UL>
```

Password Expiry

When using SYSUAF authentication it is possible for a password to pre-expired, or when a password lifetime is set for a password to expire and require respecification. By default an expired password cannot be used for access. This may be overridden using the following global configuration directive.

```
[AuthSYSUAFacceptExpPwd]  enabled
```

Expired passwords may be specially processed by specifying a URL with WASD_CONFIG_GLOBAL [AuthSysUafPwdExpURL] configuration directive (see “WASD Web Services - Install and Config”).

The WASD_CONFIG_MAP *set auth=sysuaf=pwdexpurl=<string>* rule allows the same URL to be specified on a per-path basis. When this is set a request requiring SYSUAF authentication that specifies a username with an expired password is redirected to the specified URL. This should directly or via an explanatory (wrapper) page redirect to the password change path described above. The password change dialog will have a small note indicating the password has expired and allows it to be changed.

The following WASD_CONFIG_GLOBAL directive

```
# WASD_CONFIG_GLOBAL
[AuthSysUafPwdExpURL]  https:///httpd/-/change/

# WASD_CONFIG_AUTH
[WASD_VMS_ID=id;WASD_VMS_RW=id]
/httpd/-/change/* r+w
```

would allow expired passwords to be changed.

It is also possible to redirect an expired password to a site-specific page for input and change. This allows some customization of the language and content of the expired password change dialog. An example document is provided at WASD_ROOT:[EXAMPLE]EXPIRED.SHTML ready for relocation and customisation. Due to the complexities of passing realm information and then submitting that information to the server-internal change facility some dynamic processing is required via an SSI document.

This example assumes the site-specific document has been located at WEB:[000000]EXPIRED.SHTML and is accessed using SSL.

```
# WASD_CONFIG_GLOBAL
[AuthSysUafPwdExpURL]  https:///web/expired.shtml?httpd=ignore&realm=vms

# WASD_CONFIG_AUTH
[WASD_VMS_ID=id;WASD_VMS_RW=id]
/httpd/-/change/vms/* r+w
/web/expired.shtml r+w
```

3.16 Cancelling Authorization

The reason authorization information is not required to be reentered on subsequent accesses to controlled paths is cached information the browser maintains. It is sometimes desirable to be able to access the same path using different authentication credentials, and correspondingly it would be useful if a browser had a *purge authorization cache* button, but this is commonly not the case. To provide this functionality the server must be used to "trick" the browser into cancelling the authorization information for a particular path.

This is achieved by adding a specific query string to the path requiring cancellation. The server detects this and returns an authorization failure status (401) regardless of the contents of request "Authorization:" field. This results in the browser flushing that path from the authorization cache, effectively requiring new authorization information the next time that path is accessed.

There are two variations on this mechanism.

1. The basic procedure is as follows:
   - Add the query string "?httpd=logout" to the path in question (if there is an existing query then replace it), as in the following example.

     ```
     /the/current/path?httpd=logout
     ```

   - The browser will respond with an authorization failure, and prompting to retry or reenter the username and password.
   - It is necessary to clear at least the password (i.e. remove any password from the appropriate field) and reenter.
   - The browser again responds with an authorization failure.
   - At this stage the authorization dialog can be cancelled, resulting in a server authorization failure message.
   - The original path can now be returned to and reaccessed. The browser should again prompt for authorization information at which point different credentials may be supplied.
2. A little more functional, if using a revalidation period via [AuthRevalidateUserMinutes] or 'SET auth=revalidate=' (perhaps set to something like 23:59:00, or one day), when the logout query string is supplied the server resets the entry forcing any future access to

require revalidation. A successful logout message is then generated, circumventing the need for the username/password dialog described above.

- Add or replace the query string "?httpd=logout" to the path in question as in the following example.

```
/the/current/path?httpd=logout
```

- The browser will respond with a message stating that authentication has been cancelled. That's it!

Also when using logout with a revalidation period a redirection URL may be appended to the logout query string. It then redirects to the supplied URL. It is important that the redirection is returned to the browser and not handled internally by WASD. Normal WASD redirection functionality applies.

```
?httpd=logout&goto=///
?httpd=logout&goto=///help/logout.html
?httpd=logout&goto=http://the.host.name/
```

These examples redirect to

the local home page
a specific local page
a specific remote server

respectively.

Authentication Cache

User revalidation relies on an entry being maintained in the authentication cache. Each time the entry is flushed, for whatever reason (cache congestion, command-line purge, server restart, etc.), the user will be prompted for credentials. It may be necessary to increase the size of the cache by adjusting [AuthCacheEntriesMax].

Chapter 4

Transport Layer Security

Transport Layer Security (TLS), and its predecessor **Secure Sockets Layer** (SSL), are cryptographic protocols designed to provide communication privacy over a network, in the case of HTTP between the browser (client) and the server. It also authenticates server and optionally client identity. TLS/SSL operates by establishing an encrypted communication path between the two applications, "wrapping" the entire application protocol inside the secure link, providing complete privacy for the entire transaction. In this way security-related data such as user identification and password, as well as sensitive transaction information can be protected from unauthorized access while in transit. This section is not a tutorial on TLS/SSL. It contains only information relating to WASD's use of it. See Section 4.7 for further information on TLS/SSL technology.

TLS and SSL

The terms are used interchangably in this document to represent cryptographic communication technology. They are similar but with important differences. TLS is the more modern and considered the more secure. The term SSL is still in common usage though and retained here even if (by default) WASD now only implements TLS. (When OpenSSL(.org) considers changing its name WASD will toss out the term SSL :-)

WASD implements SSL using a freely available software toolkit supported by the **OpenSSL Project**. OpenSSL licensing allows unrestricted commercial and non-commercial use. This toolkit is in use regardless of whether the WASD OpenSSL package, HP SSL for OpenVMS product, or other stand-alone OpenSSL environment is installed. It is always preferable to move to the latest support release of OpenSSL as known bugs in previous versions are progressively addressed (ignoring the issue of new bugs being introduced ;-)

TLS functionality is not supplied with the basic WASD package

In part this is due to the relative bulk of this component, in further part that the updates to each are not necessarily coincident, and also considers potential patent issues and export restrictions on some cryptography technology in some jurisdictions.

Cryptography Software

Be aware that export/import and/or use of cryptography software, or even just providing cryptography hooks, is illegal in some parts of the world. When you re-distribute this package or even email patches/suggestions to the author or other people, please **PAY CLOSE ATTENTION TO ANY APPLICABLE EXPORT/IMPORT LAWS**. The author of this package is not liable for any violations you make here.

Some Thoughts From R. S. Engelschall

Ralf S. Engelschall (rse@engelschall.com) is the author of the popular Apache *mod_ssl* package. This section is taken from the *mod_ssl* read-me and is well-worth some consideration for this and software security issues in general.

" You should be very sensible when using cryptography software, because just running an SSL server DOES NOT mean your system is then secure! This is for a number of reasons. The following questions illustrate some of the problems.

- SSL itself may not be secure. People think it is, do you?
- Does this code implement SSL correctly?
- Have the authors of the various components put in back doors?
- Does the code take appropriate measures to keep private keys private? To what extent is your cooperation in this process required?
- Is your system physically secure?
- Is your system appropriately secured from intrusion over the network?
- Whom do you trust? Do you understand the trust relationship involved in SSL certificates? Do your system administrators?
- Are your keys, and keys you trust, generated careful[ly] enough to avoid reverse engineering of the private keys?
- How do you obtain certificates, keys, and the like, securely?
- Can you trust your users to safeguard their private keys?
- Can you trust your browser to safeguard its generated private key?

" If you can't answer these questions to your personal satisfaction, then you usually have a problem. Even if you can, you may still NOT be secure. Don't blame the authors if it all goes horribly wrong. Use it at your own risk! "

4.1 SSL Functionality Sources

Secure Sockets Layer functionality is easily integrated into WASD and is available from one (or more) of the following sources. See Section 4.2 for the basics of installing WASD SSL and Section 4.3 for configuration of various aspects.

1. The **HP SSL1 for OpenVMS** product

   Information regarding HP SSL1 (Secure Sockets Layer) for OpenVMS **may** be found somewhere within this URL: http://h71000.www7.hp.com/openvms/security.html (though it's been a bit of a moving target).
   Perhaps here: http://h71000.www7.hp.com/openvms/security.html#ssl
   Perhaps from the OpenVMS (top-level) documentation URL: http://www.hp.com/go/openvms/doc

   This is provided from the directory SYS$COMMON:[SSL1] containing shared libraries, executables and templates for certificate management, etc. If this product is installed and started the WASD installation and update procedures should detect it and provide the option of compiling and/or linking WASD against its shareable libraries.

2. As a separate, easily integrated **WASD OpenSSL package**, with OpenSSL object libraries, OpenSSL utility object modules for building executables and WASD support files. It requires no compilation, only linking, and is available for Alpha, Itanium and VAX for VMS version 7.0 up to current. Obtain these from the same source as the main package.

   WASD OpenSSL installation creates an OpenSSL directory in the source area, WASD_ROOT:[SRC.OPENSSL-n_n_n], containing the OpenSSL copyright notice, object libraries, object modules for building executables, example certificates, and some other support files and documentation.

3. Using a locally compiled and installed **OpenSSL toolkit**.

4.2 WASD SSL Quick-Start

SSL functionality can be installed with a new package, or with an update, or it can be added to an existing non-SSL enabled site. The following steps give a quick outline for support of SSL.

1. If using the HP SSL1 for OpenVMS product or an already installed OpenSSL toolkit go directly to step 2. To install the WASD OpenSSL package the ZIP archive needs to be restored.

   - The ZIP archive will contain brief installation instructions. Use the following command to read this and any other information provided.

     ```
     $ UNZIP -z device:[dir]archive.ZIP
     ```

   - Either UNZIP the WASD OpenSSL package into a new installation

     ```
     $ SET DEFAULT [.WASD_ROOT]
     $ UNZIP device:[dir]archive.ZIP
     ```

   - OR into an existing installation

     ```
     $ SET DEFAULT WASD_ROOT:[000000]
     $ UNZIP device:[dir]archive.ZIP
     ```

2. It is then necessary to build the (server and Open)SSL executables.

   - If during an original INSTALL or subsequent UPDATE of the entire package the procedures detect a suitable SSL toolkit and prompt the user whether an SSL enabled server should be built.

- To to add SSL functionality to an existing but non-SSL site just the SSL components can be built using the following procedure.

```
$ @WASD_ROOT:[INSTALL]UPDATE SSL
```

3. Once linked the UPDATE.COM procedure will prompt for permission to execute the demonstration/check procedure.

   It is also possible to check the SSL package at any other time using the server demonstration procedure. It is necessary to specify that it is to use the SSL executable. Follow the displayed instructions.

```
$ @WASD_ROOT:[INSTALL]DEMO.COM SSL
```

4. Modification of server startup procedures should not be necessary. If an SSL image is detected during startup it will be used in preference to the standard image.

5. Modify the WASD_CONFIG_SERVICE configuration file to specify an SSL service. For example the following adds a generic SSL service on port 443.

```
[[https://*:443]]
```

6. Shutdown the server completely, then restart.

```
$ HTTPD /DO=EXIT
$ @WASD_ROOT:[STARTUP]STARTUP
```

7. To check the functionality (on default ports) access the server via

   Standard HTTP

```
http://the.example.com/
```

   SSL HTTP

```
https://the.example.com/
```

8. Once the server has been proved functional with the example certificate it is recommended that a server-specific certificate be created using the tools described in Section 4.4. This may then be used by placing it in the appropriate local directory, assigning the WASD_SSL_CERT symbol appropriately before startup.

4.3 SSL Configuration

The example server startup procedure already contains support for the SSL executable. If this has been used as the basis for startup then an SSL executable will be started automatically, rather than the standard executable. The SSL executable supports both standard HTTP services (ports) and HTTPS services (ports). These must be configured using the [service] parameter. SSL services are distinguished by specifying “https:” in the parameter. The default port for an SSL service is 443.

WASD can configure services using the WASD_CONFIG_GLOBAL [SSL..] directives, the per-service WASD_CONFIG_SERVICE [ServiceSSL..] directives, or the /SSL= qualifier. Configuration precedence is WASD_CONFIG_SERVICE, /SSL= and finally WASD_CONFIG_GLOBAL.

4.3.1 WASD_CONFIG_SERVICE

SSL service configuration using the WASD_CONFIG_SERVICE configuration is slightly simpler, with a specific configuration directive for each aspect. (see “WASD Web Services - Install and Config”). This example illustrates configuring the same services as used in the previous section.

```
[[http://alpha.example.com:80]]

[[https://alpha.example.com:443]]
[ServiceSSLversion]  TLSvALL
[ServiceSSLcert]  WASD_ROOT:[local]alpha.pem

[[https://beta.example.com:443]]
[ServiceSSLversion]  SSLv3
[ServiceSSLcert]  WASD_ROOT:[local]beta.pem
```

4.3.2 SSL Versions

As WASD uses the OpenSSL package in one distribution or another it largely supports all of the capability of that underlying package. The obsolete SSLv2, and the deprecated SSLv3 are no longer accepted by default. WASD default comprise the TLS family of protocols, at the time of writing, **TLSv1, TLSv1.1 and TLSv1.2**.

Some older clients employing SSLv3 may fail. Symptoms are dropped connection establishment and WATCH [x]SSL variously showing “SSL routines SSL*n*_GET_RECORD wrong version number”, “SSL routines SSL*n*_GET_CLIENT_HELLO unknown protocol”, possibly others. It is generally considered SSL best-practice not to have SSLv3 enabled but if required may be supported by configuring WASD_CONFIG_GLOBAL [SSLversion] with “SSLv3,TLSvALL”, the per-service WASD_CONFIG_SERVICE equivalent, or using the /SSL=(SSLv3,TLSvALL) command line parameter during server startup.

4.3.3 SSL Ciphers

Ciphers are the algorithms, designed and implemented on mathematical computations, that render the readable plaintext into unreadable ciphertext. Ciphers tend to be available in suites (or families) where variants, usually based on key size and therefore resistence to decryption without a known key, that browsers and otheragents negotiate on and accept when setting up a secure (encrypted) network transports with servers.

Cipher selection is important to the overall security of the supported environment as well as the range of clients and servers that can establish communication due to shared cipher suites. Including only more recent (and technically secure) ciphers can preclude older clients from establishing secure connection, and including older (and perhaps more susceptible to modern attack) ciphers increases site vunerability. Some environments, for example HTTP/2, are quite prescriptive regarding the secure connection, to the point of blacklisting protocol versions and cipher suites no longer considered secure enough.

Fortunately a number of sites provide cipher guidelines based on requirements. The Mozilla Developer Network provides these amongst other useful information on security and server side TLS.

https://wiki.mozilla.org/Security/Server_Side_TLS#Recommended_configurations

WASD has a default (built-in) functional cipher list that is general in application and relevant to when it was compiled. This in particular and site cipher lists in general, should be reviewed from time to time as opinions and requirements do change.

4.3.4 (Open)SSL Options

The OpenSSL package provides for various options to be flagged against an TLS/SSL service. WASD sets the (OpenSSL) default options and then allows these to be overwritten/set/reset using hexadecimal values representing bit patterns. OpenSSL defaults are suitable for most sites.

The SSL options directives in global and per-service configuration, and the OPTIONS= keyword for the /SSL= qualifier, accept

- 0x*XX* - overwrite the options field
- +0x*XX* - set (logical OR) the specified bit(s)
- -0x*XX* - reset (logical AND) the specified bit(s)

Alternatively, the following OpenSSL option mnemonics can be used with a leading "+" to enable, or "-" to disable

OP_ALL
OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION
OP_CIPHER_SERVER_PREFERENCE
OP_LEGACY_SERVER_CONNECT
OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION
OP_NO_TICKET
OP_SINGLE_DH_USE
OP_TLS_ROLLBACK_BUG

4.3.5 Forward Secrecy

Forward secrecy, sometimes known as perfect forward secrecy (PFS), is a property of key-agreement protocols ensuring that a session key derived from a set of long-term keys cannot be compromised if one of the long-term keys is compromised in the future.

http://en.wikipedia.org/wiki/Forward_secrecy

OpenSSL supports forward secrecy using Diffie-Hellman key exchange with elliptic curve cryptography and this relies on generating emphemeral keys based on unique, safe prime numbers. These are expensive to generate and so this is done infrequently, often during software build or installation. In the case of WASD, to maximise flexibility, these numbers are stored in external PEM-format files, by default located in the WASD_ROOT:[LOCAL] directory. These files are only briefly accessed during server startup SSL initialisation and the content later used during network connection SSL negotiation to generate the required ephemeral keys.

PFS requires a small number of elements working in concert

- Ephemeral key generation
- Selection and ordering of server ciphers
- Ensuring the server determines the cipher used (+OP_CIPHER_SERVER_PREFERENCE)

The detail is described in these references

https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy
https://community.qualys.com/blogs/securitylabs/2013/08/05/configuring-apache-nginx-and-openssl-for-forward-secrecy

NOTE

Ephemeral keys are supported beginning with WASD v10.4.1.

Executing the WASD OpenSSL procedure

```
$ @CREATE_EPHEMERAL_DH_PARAM
```

will generate site-unique files containing 512, 1024 and 2048 bit primes, and optionally copy those files to the WASD_ROOT:[LOCAL] directory. The [.CERT] directory contains files that could be used but unique, locally generated primes are preferable.

Alternatively, generated directly at the command-line using the OpenSSL *dhparam* utility, as in these examples;

```
$ openssl dhparam -out dh_param_512.pem 512
$ openssl dhparam -out dh_param_1024.pem 1024
$ openssl dhparam -out dh_param_2048.pem 2048
```

NOTE

Key generation can take some considerable time!

The file(s) must be located in the WASD_ROOT:[LOCAL] directory and the file names use the format DH_PARAM_*number-of-bits*.PEM

Alternatively, files containing emphemeral keys generated freshly with each release, may be copied from the WASD OpenSSL package using

```
$ COPY WASD_ROOT:[SRC.OPENSSL-n_n_n.WASD.CERT]DH_PARAM_*.PEM WASD_ROOT:[LOCAL]
```

4.3.6 Session Resumption

When a TLS/SSL connection is initiated an expensive handshake (in terms of time and compute) is required to establish the cryptographic and other elements of the connection. Mitigation of this expense is undertaken by allowing the resumption of a previous session (abbreviating the handshake exchanges) using connection state stored either at the server or at the client.

- **Session Ticket**

  This TLS extension provides the connection state to the client, encrypted with keys available only to the server. The client stores the (encrypted) state and when (re-)connecting to the server provides that ticket in the initial part of the handshake. The server decrypts the ticket and if valid expedites the connection by resuming the previously negotiated session. This is the more modern, almost universally supported mechanism and is generally enabled by default.

Session tickets introduce a potential vulnerability to TLS security, in particular to the benefits of Forward Secrecy (PFS). If the ticket can be compromised, through theft of the keys or brute-force decryption attack, the entire session becomes vulnerable to attack. It is therefore advised to periodically rotate (change) the keys used by the server to encrypt the tickets. WASD does this every (RFC recommended) 24 hours, at midnight (local time).

Where a site is provided by multiple servers and connections distributed between these, session resumption using tickets relies on each server using the same keys. The current keys must be distributed to each server (using a secure mechanism) and this performed every time the keys are rotated. WASD uses the DLM to perform this for multiple per-node and cluster-wide instances as applicable.

- **Session ID**

  In a full handshake the server sends a Session ID (unique, non-repeating value) as part of the handshake. On a subsequent connection the client can pass this session ID back to the server when connecting. To support session resumption via session IDs the server must maintain a cache that maps past session IDs to those sessions' states. The cache has limited capacity and is expensive for the server to maintain. If the session ID is still available in the cache the session can be resumed. This is the original session resumption mechanism.

  Where a single WASD instance is involved the session cache is implemented in-memory. With multiple instances on a single node it is provided across those instances using a shared global section. The cacpacity of this shared cache is determined by the WASD_CONFIG_GLOBAL directives [SSLinstanceCacheMax] and [SSLinstanceCacheSize] directives. There is no cluster-wide session cache. When multiple instances are in use the shared session cache is enabled by default. Session ID caching may be globally disabled by setting [SSLsessionCacheMax] to -1.

With Session Tickets being the more modern, flexible and efficient solution to session resumption (and being available cluster-wide) it is recommended that WASD sites disable Session ID caching.

The default maximum period for session reuse is five minutes. This may be set globally using the [SSLsessionLifetime] directive or on a per-service basis using [ServiceSSLsession-Lifetime].

To some extent, the relatively long-lived connections and lower concurrency with HTTP/2 means the importance of session resumption in improving request latency and connection overhead is reduced.

4.3.7 Strict Transport Security

HTTP Strict Transport Security (HSTS) is a security policy mechanism which helps protect sites against protocol downgrade attack and cookie hijacking. It allows web servers to declare that browsers and other complying agents should only interact using secure (TLS) HTTP connections and never via clear-text HTTP. HSTS is an IETF standard specified in RFC 6797.

When global configuration directive [SSLstrictTransSec] is non-zero, or per-service configuration directive [ServiceSSLstrictTransSec] is non-zero, or a path is *SET response=sts=<value>*, TLS/SSL HTTP responses include a "Strict-Transport-Security: max-age=*seconds*" header field. Conforming agents note this period and refuse to communicate with the site via cleartext HTTP for the period represented by the integer number of seconds specified.

4.3.8 SSL Server Certificate

The server certificate is used by the browser to authenticate the server against the server certificate Certificate Authority (CA), in making a secure connection, and in establishing a trust relationship between the browser and server. By default this is located using the WASD_CONFIG_GLOBAL [SSLcert] or WASD_CONFIG_SERVICE [ServiceSSLcert] configuration directive, the WASD_CONFIG_SSL_CERT logical name, or using the /SSL= command-line qualifier, however if required. Each SSL service can have an individual certificate configured as in the example above.

4.3.9 SSL Private Key

The *private key* is used to validate and enable the server certificate. A private key is enabled using a *secret*, a password. It is common practice to embed this (encrypted) password within the private key data. This private key can be appended to the server certificate file, or it can be supplied separately. If provided separately it can be located using the WASD_CONFIG_GLOBAL [SSLkey] or WASD_CONFIG_SERVICE [ServiceSSLkey] configuration directive, tor using the WASD_CONFIG_SSL_KEY logical. When the password is embedded in the private key information it becomes vulnerable to being stolen as an enabled key. For this reason it is possible to provide the password separately and manually.

If the password key is not found with the key during startup the server will request that it be entered at the command-line. This request is made via the HTTPDMON "STATUS:" line (see "WASD Web Services - Install and Config"), and if any OPCOM category is enabled via an operator message. If the private key password is not available with the key it is recommended that OPCOM be configured, enabled and monitored at all times.

When a private key password is requested by the server it is supplied using the /DO=SSL=KEY=PASSWORD directive (Section 9.7). This must be used at the command line on the same system as the server is executing. The server then prompts for the password.

```
Enter private key password []:
```

The password is not echoed. When entered the password is securely supplied to the server and startup progresses. An incorrect password will be reprompted for twice (i.e. up to three attempts are allowed) before the startup continues with the particular service not configured and unavailable. Entering a password consisting of all spaces will cause the server to abort the full startup and exit from the system.

4.3.10 SSL Virtual Services

Multiple virtual SSL services (https:) sharing the same or individual certificates (and other characteristics) can essentially be configured against any host name (unique IP address or host name alias) and/or port in the same way as standard services (http:).

WASD SSL implements **Server Name Indication** (SNI), an extension to the TLS protocol that indicates what hostname the client is attempting to connect to at the start of the handshaking process. This allows a server to present multiple certificates on the same IP address and port number and hence allows multiple secure (HTTPS) websites (or any other Service over TLS) to be served off the same IP address without requiring all those sites to use the same certificate.

When the client presents an SNI server name during SSL connection establishment, WASD searches the list of services it is offering for an SSL service (the first hit) operating with a name matching the SNI server name. If matched, the SSL context (certificate, etc.) of that service is used to establish the connection. If not matched, the service the TCP/IP connection originally arrived at is used.

4.3.11 SSL Access Control

When authorization is in place (Chapter 3) access to username/password controlled data/functionality benefits enormously from the privacy of an authorization environment inherently secured via the encrypted communications of SSL. In addition there is the possibility of authentication via client X.509 certification (Section 4.3.12). SSL may be used as part of the site's access control policy, as whole-of-site, see Section 3.2, or on a per-path basis (see "WASD Web Services - Install and Config").

4.3.12 Authorization Using X.509 Certification

The server access control functionality (authentication and authorization) allows the use of *public key infrastructure* (PKI) X.509 v3 client certificates for establishing identity and based on that apply authorization constraints. See Chapter 3 for general information on WASD authorization and Section 3.4 for configuring a X509 realm. Section 4.7 provides introductory references on public-key cryptography and PKI.

A client certificate is stored by the browser. During an SSL transaction the server can request that such a certificate be provided. For the initial instance of such a request the browser activates a dialog requesting the user select one of any certificates it has installed. If selected it is transmitted securely to the server which will usually (though optionally not) authenticate its Certificate Authority to establish its integrity. If accepted it can then be used as an authenticated identity. This obviates the use of username/password dialogs.

Important

Neither username/password nor certificate-based authentication addresses security issues related to access to individual machines and stored certificates, or to password confidentiality. Public-key cryptography only verifies that a private key used to sign some data corresponds to the public key in a certificate. It is a user responsibility to protect a machine's physical security and to keep private-key passwords secret.

The initial negotiation and verification of a client certificate is a relatively resource intensive process. Once established however, OpenSSL sessions are usually either stored in a cache or stored encrypted withing the client, reducing subsequent request overheads significantly. Each session has a specified expiry period after which the client is forced to negotiate a new session. This period is adjustable using the "[LT:integer]" and "[TO:integer]" directives described below.

4.3.13 X.509 Certificate Renegotiation

An X.509 client certificate is requested at either TLS/SSL connection establishment (WASD_CONFIG_GLOBAL [SSLverifyPeer], WASD_CONFIG_SERVICE [ServiceSSLverifyPeer]) or once the request has been made and assessed against authorisation rules. If an X509 realm controls access to the resources then the TLS/SSL connection is queried for an X.509 client certificate to authenticate the client and authorise the access.

This is performed via a TLS/SSL renegotiation and for this the connection must have been cleared of request data. In the case of a HEAD, GET, OPTIONS, etc. request, this already has implicitly occurred by there being no request body. For POST, PROPFIND, PUT, etc. requests, the client most likely already will be transmitting the request body. This (*application data*) must be absorbed before the client certificate renegotiation can be performed.

In avoiding disruption to the current request, any request body must be buffered (in full, based on the content length specified in the header) before issuing the renegotiation. This consumes memory and potentially large quantities. The default maximum buffer space is 1MB. The maximum request body size and hence maximum memory accomodated can be configured using the per-service WASD_CONFIG_SERVICE [ServiceSSLverifyDataMax] directive, or the global WASD_CONFIG_GLOBAL configuration directive [SSLverifyDataMax].

Where a request with a body exceeds the maximum allowed buffer space the authorisation fails. This can be observed using WATCH. Where very large files are being sent the only solution is to first authenticate with a request without a body (e.g. using OPTIONS) then using the persistent connection and associated X.509 authentication perform the PUT or POST.

4.3.14 Features

WASD provides a range of capabilities when using X.509 client certificates.

- **By Service -** all SSL connections to such a service will be requested to supply a client certificate during the initial SSL handshake. This is more efficient than requesting later in the transaction, as happens with per-resource authorization. A client cannot connect successfully to this type of service without supplying an acceptable certificate.
- **By Resource -** using authorization rules in the WASD_CONFIG_AUTH file specifying a path against an [X509] realm causes the server to suspend request processing and renegotiate with the client to supply a certificate. If a suitable certificate is supplied the request authorization continues with normal processing. This obviously incurs an additional network transaction.

- **Optional access control -** once an acceptable certificate is supplied it can be subject to further access control by matching against its contents. The *Issuer* (CA) and the *Subject* (client) *Distinguished Name* (DN) has various components including the name of the organization providing the certificate (e.g. “VeriSign”, “Thawte”), location, common name, email address, etc. Those certificates matching or not matching the parameters are allowed or denied access.
- **Certificate verification -** by default supplied certificates have their CA verified by comparing to a list of recognised CA certificates stored in a server configuration file. If the CA component of the client certificate cannot be verified the connection is terminated before the HTTP request can begin. Although this is obviously required behaviour for authentication there may be other circumstances where verification is not required, a certificate content display service for instance. WASD optionally allows non-verified certificates to be used on a per-resource basis.
- **“Fingerprint” REMOTE_USER -** when a certificate is accepted by the server it generates a unique *fingerprint* of the certificate. By default, this 32 digit hexadecimal number is used by the server as an *effective username*, one that would normally be supplied via a username/password dialog (as an alternative see the section immediately below). This effective username becomes that available via the CGI variable REMOTE_USER. Although a 32 digit number is not particularly site-administrator friendly it is a unique representation (MD5 digest) of the individual certificate and can be used in WASD_CONFIG_AUTH access-restriction directives and included in group lists and databases for full WASD authorization control.
- **CN/DN record REMOTE_USER -** provides an alternative to using a “fingerprint” REMOTE_USER. Using the [RU:/*record*=] conditional (see below) is becomes possible to specify that the remote-user string be obtained from the specified record of the client certificate subject field. Note that there is a (fairly generous) size limitation on the user name and that any white-space in such a record is converted to underscores. Although any record can be used the more obvious candidates are /O=, /OU=, /CN=, /S=, /UID= and /EMAIL=. Note that (even with the default CA verfication) the certificate CAs that this is possible against should be further constrained through the use of a [IS:/*record*=*string*] conditional (see example below).
- **Subject Alternative Name REMOTE_USER -** a common X509 V3 extension for providing identifying data in a certificate, can also be used to derive the remote user string.
- **X509 extension REMOTE_USER -** the content of any other extension field suitably filtered.

4.3.15 Subject Alternative Name and Other Extensions

The basic syntax for this field is the full extension name, and the short-hand equivalent.

```
[X509]
/VMS/* r+w,param="[ru:X509v3_subject_Alternative_Name]"
/VMS/* r+w,param="[ru:X509v3_SAN]"
```

The Subject Alternative Name (SAN) extension (in common with many others) may contain multiple data elements, each with a leading name, a colon, and a (if multi line) carriage-control terminated value. WASD parses these into unqiue fields using keywords fixed in function SesolaCertKeyword( ) and the site configurable logical name WASD_X509_EXTENSION_KEYWORDS value. To select one of these fields, for example the common (Microsoft) user principal name (UPN), append the required field name to the extension name as shown in the following example (includes "shorthand" equivalents, along with the underscore and equate variants). Note that the identifying name match is not case sensitive.

```
[X509]
/VMS/* r+w,param="[ru:X509V3_Subject_Alternative_Name_UserPrincipalName]"
/VMS/* r+w,param="[ru:X509V3_Subject_Alternative_Name=UserPrincipalName]"
/VMS/* r+w,param="[ru:X509v3_SAN_UPN]"
/VMS/* r+w,param="[ru:X509v3_SAN=UPN]"
/VMS/* r+w,param="[ru:X509V3_Subject_Alternative_Name_rfc822Name]"
/VMS/* r+w,param="[ru:X509V3_Subject_Alternative_Name=rfc822Name]"
/VMS/* r+w,param="[ru:X509v3_SAN_822]"
/VMS/* r+w,param="[ru:X509v3_SAN=822]"
```

Object Identifiers (OIDs) may be used for either record and field name (if an unknown otherName) by prefixing with “OID_”. For example, the SAN may be alternatively selected, and the (Microsoft) UPN, as in the following examples.

```
/VMS/* r+w,param="[ru:OID_2_5_29_17]"
/VMS/* r+w,param="[ru:OID_2_5_29_17_UPN]"
/VMS/* r+w,param="[ru:OID_2_5_29_17=UPN]"
/VMS/* r+w,param="[ru:X509v3_SAN_OID_1_3_6_1_20_2_3]"
/VMS/* r+w,param="[ru:X509v3_SAN_OID=1_3_6_1_20_2_3]"
```

Extension Visibility

X509 certificate extensions are in general visible from WATCH and accessible via CGI variables (when enabled using SET *SSLCGI=apache_mod_ssl_extens* and *SSLCGI=apache_mod_ssl_client* path mappings). The identifying names derived from X509 extensions are built of the alphanumerics in the element names. Non-alphanumerics (e.g. spaces) have underscores substituted. Multiple underscores are compressed into singles. Where elements have identical names the first multiple has TWO underscores and the digit two appended, the second mutiple, two underscores and three appended, etc.

4.3.16 X509 Configuration

Of course, the WASD OpenSSL component must be installed and in use to apply client X.509 certificate authorization. There is general server setup, then per-service and per-resource configuration.

General Setup

Client certificate authorization has reasonable defaults. If some aspect requires site refinement the WASD_CONFIG_GLOBAL [SSL..] directives (see “WASD Web Services - Install and Config”) or command-line /SSL= qualifier parameters can provide per-server defaults.

- (CACHE=integer) sets the session size (128 entries by default)

- (CAFILE=file-name) sets the location of the CA verification store file (also can be set via WASD_CONFIG_SSL_CAFILE logical).
- (TIMEOUT=integer) sets the session expiry period in minutes (5 by default)
- (VERIFY=integer) sets the depth to which client certificate CAs are verified (default is 10)

The location of the CA verification file can also be determined using the logical name WASD_CONFIG_SSL_CAFILE. The order of precedence for using these specifications is

1. per-service configuration using WASD_CONFIG_SERVICE or WASD_CONFIG_GLOBAL
2. per-server using /SSL=CAFILE=filename
3. per-server using WASD_CONFIG_SSL_CAFILE

By Service

The WASD_CONFIG_SERVICE directive is provided for per-service CA file specification, if necessary allowing different services to accept a different mix of CAs.

```
[[https://the.example.com:443]]
[ServiceSSLVerifyPeer]  enabled
[ServiceSSLVerifyPeerCAfile]  WASD_ROOT:[LOCAL]CA_THE_HOST_NAME.TXT
```

By Resource

Client certificate authorization is probably most usefully applied on a per-resource (per-request-path) basis using WASD_CONFIG_AUTH configuration file rules. Of course, per-resource control also applies to services that always require a client certificate (the only difference is the certificate has already been negotiated for during the initial connection handshake). The reserved realm name “X509” activates client certificate authentication when a rule belonging to that realm is triggered. The following example shows such a rule providing read access to those possessing any verified certificate.

```
[X509]
/path/requiring/cert/* r
```

Optional directives may be supplied to the X.509 authenticator controlling what mode the certificate is accepted in, as well a further access-restriction rules on specifically which certificates may or may not be accepted for authorization. Such directives are passed via the “param=” mechanism. The following real-life example shows a script path requiring a mandatory certificate, but not necessarily having the CA verified. This would allow a certificate display service to be established, the “[to:EXPIRED]” directive forcing the client to explicitly select a certificate with each access.

```
[X509]
/cgi-bin/client_cert_details r,param="[vf:OPTIONAL][to:EXPIRED]"
```

A number of such directives are available controlling some aspects of the certificate negotiation and verification. The “[LT:integer]” directive causes a verified certificate selection to continue to be valid for the specified period as long as requests continue during that period (lifetime is reset with each access).

- [DP:integer] verify certificate CA chain to this depth (default 10)

- [LT:integer] verified certificate lifetime in minutes (disabled by default)
- [RU:/record=] derive the remote-user name from the specified certificate subject field DN record
- [TO:integer] session cache entry timeout in minutes (default 5)
- [TO:EXPIRED] session cache entry is forced to expire (initating renegotiation)
- [VF:NONE] no certificate is required (any existing is cancelled)
- [VF:OPTIONAL] certificate is required, CA verification is not required
- [VF:REQUIRED] the certificate must pass CA verification (the default)

Optional "param=" passed conditionals may also be used to provide additional filtering on which certificates may or may not be used against the particular path. This is based on pattern matching against client certificate components.

- [CI:string] transaction cipher
- [IS:/record=string] specified Issuer (CA) DN record only
- [IS:string] entire Issuer (CA) DN
- [KS:integer] minimum key size
- [SU:/record=string] specified Subject (client) DN record only
- [SU:string] entire Subject (client) DN

These function and can be used in a similar fashion to mapping rule conditionals (see "WASD Web Services - Install and Config" document, "Conditional Configuration" section). This includes the logical ORing, ANDing and negating of conditionals. Asterisk wildcards match any zero or more characters, percent characters any single character. Matching is case-insensitive.

Note that the "IS:" and "SU:" conditionals each have a *specific-record* and an *entire-field* mode. If the conditional string begins with a slash then it is considered to be a match against a specified record contents within the field. If it begins with a wildcard then it is matched against the entire field contents. Certificate DN records recognised by WASD,

/C= countryName
/ST= stateOrProvinceName
/SP= stateOrProvinceName
/L= localityName
/O= organizationName
/OU= organizationalUnitName
/CN= commonName
/T= title
/I= initials
/G= givenName
/S= surname
/D= description
/UID= uniqueIdentifier
/Email= pkcs9_emailAddress

The following (fairly contrived) examples provide an illustration of the basics of X509 conditionals. When matching against Issuer and Subject DNs some knowlege of their contents and structure is required (see Section 4.7 for some basic resources).

```
[X509]
# only give "VeriSign"ed ones access
/controlled/path1/* r+w,param="[IS:/O=VeriSign\ Inc.]"
# only give non-"VeriSign"ed ones access
/controlled/path2/* r+w,param="[!IS:/O=VeriSign\ Inc.]"
# only allow 128 bit keys using RC4-MD5 access
/controlled/path3/* r+w,param="[KS:128][CI:RC4-MD5]"
# only give a "Thawte"-signed client based in Australia
# with the following email address access
/controlled/path4/* r+w,param="\
[IS:*/O=Thawte\ Consulting\ cc/*]\
[SU:*/C=AU/*/Email=mark.daniel@wasd.vsm.com.au*]"
# use the subject DN common-name record as the remote-user name
# furthermore, restrict the CA's allowed to be used this way
/VMS/* r+w,param="[RU:/CN=][IS:/O=WASD\ CA\ Cert]"
```

Of course, access control via group membership is also available. The *effective username* for the list is the 32 digit fingerprint of the client certificate (shown as REMOTE_USER IN the first example of Section 4.3.18), or the Subject DN record as specified using the [RU:/*record*=] directive. This may be entered into simple lists as part of a group of which membership then controls access to the resource. The following examples show the contents of simple list files containing the X.509 fingerprints, derived remote-user names, and the required WASD_CONFIG_AUTH realm entries.

```
# FINGERPRINTS.$HTL
# (a file of X.509 fingerprints for access to "/path/requiring/cert/")
106C8342890A1703AAA517317B145BF7  mark.daniel@wasd.vsm.com.au
6ADA07108C20338ADDC3613D6D8B159D  just.another@where.ever.com

# CERT_CN.$HTL
# (a file of X.509 remote-user names derived using [RU:/CN=]
Mark_Daniel mark.daniel@wasd.vsm.com.au
Just_Another just.another@where.ever.com

[X509;FINGERPRINTS=list]
/path/requiring/cert/* r+w

[X509;CERT_CN=list]
/path/requiring/cn/* r+w
```

In a similar fashion the effective username can be placed in an access restriction list. The following configuration would only allow the user of the certificate access to the specified resources. Other verified certificate holders would be denied access.

```
[X509]
/httpd/-/admin/* ~106C8342890A1703AAA517317B145BF7,r+w
/wasd_root/local/* ~106C8342890A1703AAA517317B145BF7,r+w

/other/path/* ~Mark_Daniel,r+w,param="[ru:/cn=]"
/yet/another/path/* ~Just_Another,r+w,param="[ru:/cn=]"
```

4.3.17 Certificate Authority Verification File

For the CA certificate component of the client certificate to be verified as being what it claims to be (and thus establishing the integrity of the client certificate) a list of such certificates must be provided for comparison purposes. For WASD this list is contained in a single, plain-text file variously specified using either the WASD_CONFIG_SSL_CAFILE logical or per-service "[ServiceSSLclientCAfile]" directives, or the global [SSLverifyPeerCAFile] directive.

Copies of CA certificates are available for such purposes. The PEM copies (base-64 encoded versions of the binary certificate) can be placed into this file using any desired text editor. Comments may be inserted by prefixing with the "#" character. For WASD this would be best stored in the WASD_ROOT:[LOCAL] directory, or site equivalent.

An example of how such a file appears is provided below (bulk of the file has been 8< snipped 8< for bevity).

```
##
## Bundle of CA Root Certificates
##
## Certificate data from Mozilla as of: Wed Apr 20 03:12:05 2016
##
## This is a bundle of X.509 certificates of public Certificate Authorities
## (CA). These were automatically extracted from Mozilla's root certificates
## file (certdata.txt).  This file can be found in the mozilla source tree:
## http://hg.mozilla.org/releases/mozilla-release/raw-file/default/security/nss/lib/ckfw/buil
##
## It contains the certificates in PEM format and therefore
## can be directly used with curl / libcurl / php_curl, or with
## an Apache+mod_ssl webserver for SSL client authentication.
## Just configure this file as the SSLCACertificateFile.
##
## Conversion done with mk-ca-bundle.pl version 1.25.
## SHA1: 5df367cda83086392e1acdf22bfef00c48d5eba6
##

GlobalSign Root CA
==================
-----BEGIN CERTIFICATE-----
MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkGA1UEBhMCQkUx
GTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jvb3QgQ0ExGzAZBgNVBAMTEkds
b2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAwMDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNV
BAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYD
VQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDa
DuaZjc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6arymAZavpxy0Sy6sc
THAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCOXkNz8kHp1Wrjsok6Vjk4bwY8iGlb
Kk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdGsnUOhugZitVtbNV4FpWi6cgKOOvyJBNP
c1STE4U6G7weNLWLBYy5d4ux2x8gkasJU26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrX
gzT/LCrBbBlDSgeF59N89iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
HRMBAf8EBTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0BAQUF
AAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOzyj1hTdNGCbM+w6Dj
Y1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE38NflNUVyRRBnMRddWQVDf9VMOyG
j/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymPAbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhH
hm4qxFYxldBniYUr+WymXUadDKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveC
X4XSQRjbgbMEHMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==
-----END CERTIFICATE-----
8< snip 8<
```

The WASD OpenSSL package provides an example CA verification file. The exact date and source can be found in the opening commentary of the file itself. The contents of this file easily can be pared down to the minimum certificates required for any given site.

The bundle may be refreshed at any time using any reliable source. The cURL project provides such a resource suitable for its own use, Apache mod_ssl and WASD. This is sourced from the root certificates used by the Mozilla Foundation for its Firefox product (and others). Mozilla uses a non-PEM format source which must be converted before use by WASD. The cURL site provides this already converted for use with its own utility and made available as a general resource.

http://curl.haxx.se/
http://curl.haxx.se/docs/caextract.html

Download the bundle using a command-line tool as in this example

```
$ curl -o ca-bundle_crt.txt https://curl.haxx.se/ca/cacert.pem
```

or as a save-as dialogue click from your favourite browser and then a transfer onto the VMS system.

https://curl.haxx.se/ca/cacert.pem

4.3.18 X.509 Authorization CGI Variables

CGI variables specific to client certificate authorization are always generated for use by scripts and SSI documents. These along with the general WASD authorization variables are shown in the example below. Note, that due to length of particular items some in this example are displayed wrapped.

```
WWW_AUTH_ACCESS == "READ+WRITE"
WWW_AUTH_GROUP == ""
WWW_AUTH_REALM == "X509"
WWW_AUTH_REALM_DESCRIPTION == "X509 Client Certs"
WWW_AUTH_TYPE == "X509"
WWW_AUTH_USER == "Mark Daniel, mark.daniel@wasd.vsm.com.au"
WWW_AUTH_X509_CIPHER == "RC4-MD5"
WWW_AUTH_X509_FINGERPRINT == "10:6C:83:42:89:0A:17:03:AA:A5:17:31:7B:14:5B:F7"
WWW_AUTH_X509_ISSUER == "/O=VeriSign, Inc./OU=VeriSign Trust
Network/OU=www.verisign.com/repository/RPA Incorp. By
Ref.,LIAB.LTD(c)98/CN=VeriSign Class 1 CA Individual Subscriber-Persona Not
Validated"
WWW_AUTH_X509_KEYSIZE == "128"
WWW_AUTH_X509_SUBJECT == "/O=VeriSign, Inc./OU=VeriSign Trust
Network/OU=www.verisign.com/repository/RPA Incorp. by
Ref.,LIAB.LTD(c)98/OU=Persona Not Validated/OU=Digital ID Class 1 - Netscape
/CN=Mark Daniel/Email=mark.daniel@wasd.vsm.com.au"
WWW_REMOTE_USER == "106C8342890A1703AAA517317B145BF7"
```

Other CGI variables optionally may be enabled using WASD_CONFIG_MAP mapping rules. See Section 4.5. Specific client certificate variables providing the details of such certificates are available with SSLCGI=apache_mod_ssl. These are of course in addition to the more general apache_mod_ssl variables described in the above section. Note that where some ASN.1 records are duplicated (as in SSL_CLIENT_S_DN) some variables will contain newline characters (0x10) between those elements (e.g. SSL_CLIENT_S_DN_OU). The line breaks in this example do not necesarily reflect those characters.

```
WWW_SSL_CLIENT_A_KEY == "rsaEncryption"
WWW_SSL_CLIENT_A_SIG == "md5WithRSAEncryption"
WWW_SSL_CLIENT_I_DN == "/O=VeriSign, Inc./OU=VeriSign Trust Network
/OU=www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98
/CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated"
WWW_SSL_CLIENT_I_DN_CN == "VeriSign Class 1 CA Individual Subscriber-Persona
Not Validated"
WWW_SSL_CLIENT_I_DN_O == "VeriSign, Inc."
WWW_SSL_CLIENT_I_DN_OU == "VeriSign Trust Network
www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98"
WWW_SSL_CLIENT_M_SERIAL == "0BF233D4FE232A90F3F98B2CE0D7DADA"
WWW_SSL_CLIENT_M_VERSION == "3"
WWW_SSL_CLIENT_S_DN == "/O=VeriSign, Inc./OU=VeriSign Trust Network
/OU=www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98
/OU=Persona Not Validated/OU=Digital ID Class 1 - Netscape
/CN=Mark Daniel/Email=mark.daniel@wasd.vsm.com.au"
WWW_SSL_CLIENT_S_DN_CN == "Mark Daniel"
WWW_SSL_CLIENT_S_DN_EMAIL == "mark.daniel@wasd.vsm.com.au"
WWW_SSL_CLIENT_S_DN_O == "VeriSign, Inc."
WWW_SSL_CLIENT_S_DN_OU == "VeriSign Trust Network
www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98
Persona Not Validated.Digital ID Class 1 - Netscape"
WWW_SSL_CLIENT_V_END == "Feb 10 23:59:59 2001 GMT"
WWW_SSL_CLIENT_V_START == "Dec 12 00:00:00 2000 GMT"
```

4.4 Certificate Management

This is not a tutorial on X.509 certificates and their management. Refer to the listed references, Section 4.7, for further information on this aspect. It does provide some basic guidelines.

Certificates identify something or someone, associating a public cryptographic key with the identity of the certificate holder. It includes a distinguished name, identification and signature of the certificate authority (CA, the issuer and guarantor of the certificate), and the period for which the certificate is valid, possibly with other, additional information.

The three types of certificates of interest here should not be confused.

- **CA -** The Certificate Authority identifies the *authority*, or organization, that issues a certificate.
- **Server -** Identifies a particular end-service. Its value as an guarantee of identity is founded in the *authority* of the organization that issues the certificate. It is the certificate specified to the server at startup.
- **Client -** Identifies a particular client to a server via SSL (client authentication). Typically, the identity of the client is assumed to be the same as the identity of a human being. Again, its value as an guarantee of identity is founded in the *authority* of the organization that issues the certificate.

The various OpenSSL tools are available for management of all of these certificate types in each of the three SSL environments.

- The HP SSL1 for OpenVMS product provides the “SSL Certificate Tool” procedure can be used to perform most required certificate management tasks from a menu-driven interface (HP SSL1 V1.0-2C example).

```
$ @SSL1$COM:SSL1$CERT_TOOL.COM

      SSL Certificate Tool

                              Main Menu

          1. View a Certificate
          2. View a Certificate Signing Request
          3. Create a Certificate Signing Request
          4. Create a Self-Signed Certificate
          5. Create a CA (Certification Authority) Certificate
          6. Sign a Certificate Signing Request
          7. Revoke a Certificate
          8. Create a Certificate Revocation List
          9. Hash Certificates
         10. Hash Certificate Revocations
         11. Exit

         Enter Option:
```

- The WASD OpenSSL kit provides elementary DCL procedures and brief notes in the WASD_ROOT:[SRC.OPENSSL-n_n_n.WASD] directory for some procedure-driven support of these activities.
- The standard OpenSSL toolkit provides a number of command-line tools for creation and management of X.509 certificates.

4.4.1 Server Certificate

The server uses a certificate to establish its identity during the initial phase of the SSL protocol exchange. Each server should have a unique certificate. An example certificate is provided with the WASD OpenSSL package. If this is not available (for instance when using the HP SSL1 for OpenVMS product) then the server will fallback to an internal, default certificate that allows SSL functionality even when no external certification is available. If a “live” SSL site is required a unique certificate issued by a third-party Certificate Authority is desirable.

A working alternative to obtaining one of these certificates is provided by the WASD support DCL procedures, which are quick hacks to ease the production of certificates on an ad hoc basis. In all cases it is preferable to directly use the utilities provided with OpenSSL, but the documentation tends to be rather sparse.

The first requirement may be a tailored “Certificate Authority” certificate. As the Certificate Authority is non-authoritative (not trying to be too oxymoronic, i.e. not a well-known CA) these certificates have little value except to allow SSL transactions to be established with trusting clients. More commonly “Server Certificates” for specific host names are required.

Loading Authority Certificates

CA certificates can be loaded into browsers to allow sites using that CA to be accessed by that browser without further dialog. Browsers commonly invoke a server certificate load dialog when encountering a site using a valid but unknown server certificate.

A manual load is accomplished by requesting the certificate in a format appropriate to the particular browser. This triggers a browser dialog with the user to confirm or refuse the loading of that certificate into the browser Certificate Authority database.

To facilitate loading CA certificates into a browser ensure the following entries are contained in the HTTP$CONFIG configuration file:

```
[AddIcon]
/httpd/-/binary.gif  [BIN]  application/x-x509-ca-cert

[AddType]
.CRT  application/x-x509-ca-cert  -  DER certifcate (MSIE)
.PEM  application/x-x509-ca-cert  -  Privacy Enhanced Mail certificate
```

Then just provide a link to the required certificate file(s), and click.

Changing Server Certificates

If a site's server (or CA certificate) is changed and the server restarted any executing browsers will probably complain (Netscape Navigator reports an I/O error). In this case open the browser's certificate database and delete any relevant, permanently stored certificate entry, then close and restart the browser. The next access should initiate the server certificate dialog, or the CA certificate may be explicitly reloaded.

4.4.2 Client Certificate

As with server certificates, client certficates are best obtained from a recognised Certificate Authority. However, for testing and experimental purposes WASD provides some elementary CGI scripts and DCL procedures to assist in locally generating X.509 client certificates and installing them into user browsers.

Manual Generation

The OpenSSL CA certificate generation utility can be used at the command line to process a CSR. That CSR could have been generated via an online HTML form.

Command-Line Generation

A basic DCL procedure providing the necessary Certificate Signing Request (CSR) then subsequent certificate generation and optional PKSC#12 conversion is provided by WASD_ROOT:[SRC.OPENSSL-n_n_n.WASD]CREATE_CLIENT_CERT.COM

Semi-Automatic Generation

Using this approach the user generates a Certificate Signing Request (CSR) online, which is then further processed off-line, at the discretion of the site administrator. Only Netscape browsers are supported for what is described below.

1. Provide an HTML form with the appropriate fields for each of the required ASN.1 fields used in X.509 certificates, plus a special, Netscape-sepcific one named <KEYGEN>, which allows the creation of a user's private-key. The user completes the elements of that form and when submitted the contents are emailed to the site administrator. A CSR can be freely transmitted as open text because it is secured by the private-key generated and only stored on the user's local machine.

2. The site administrator receives such a CSR by email. At that person's discretion and availability the CSR is input (cut-and-paste to eliminate errors) to a form activating a local CGI script requiring authorization for activation. The CGI script processes the CSR submitted by the form and creates using the OpenSSL CA certificate signing utility to generate a certificate (or an error if there is a problem).
3. If a client certificate is successfully generated it can either be delivered back to the user via email, for local saving and import, or made available for a short period via the Web for the user to collect (via a file with the content-type of "application/x-x509-user-cert"). Notification of such availability could be made using email.

A basic DCL procedure providing such a facility is WASD_ROOT:[SRC.OPENSSL-n_n_n.WASD]CLIENT_CERT_REQUEST.COM

This semi-automatic method would probably be the author's preference over the on-demand approach (see below).

Generation On-Demand

Automatic, on-demand client certificate generation allows any user (subject to access controls) to generate a client certificate automatically via an online service. While this may not generally be a useful thing for a site to provide there may be occasions for its use. It is a three part process. Only Netscape browsers are supported for what is described below.

1. As with the semi-automatic approach an HTML form allows a user to input and submit certificate details.
2. The submitted form activates a CGI script which collates the form details generating the Certificate Signing Request (CSR). The CSR is then used directly by the OpenSSL CA certificate signing utility to generate a certificate (or an error if there is a problem).
3. If a client certificate is successfully generated it is delivered back to the browser with a content-type of "application/x-x509-user-cert" which results in the browser installing it in its certificate database.

A basic DCL procedure providing such a facility is WASD_ROOT:[SRC.OPENSSL-n_n_n.WASD]CLIENT_CERT_REQUEST.COM (and yes, it's the same procedure as used with the semi-automatic approach, just configured differently).

4.4.3 Certificate Signing Request

Recognised Certificate Authorities (CAs) such as Thawte and VeriSign publish lists of requirements for obtaining a server certificate. These often include such documents required to prove organisational name and the right to use the domain name being requested. Check the particular vendor for the exact requirements.

In addition, a document containing the site's private key is required. This is known as the Certificate Signing Request (CSR) and must be generated digitally at the originating site.

Using the HP SSL1 for OpenVMS product "SSL Certificate Tool" described in Section 4.4 a CSR can easily be generated using its menu-driven interface. The alternative is using a command-line interface tool.

The following instructions provide the basics for generating a CSR at the command-line in the WASD and generally the any OpenSSL environment (including the HP SSL1 for OpenVMS product).

1. Change to a secure directory. The following is a suggestion.

```
$ SET DEFAULT WASD_ROOT:[LOCAL]
```

2. Assign a foreign verb for the OPENSSL application. The location may vary a little depending on which OpenSSL package you have installed.

```
$ OPENSSL == "$WASD_ROOT:[SRC.OPENSSL-version.AXP.EXE.APPS]OPENSSL.EXE"
```

   When using the HP SSL1 for OpenVMS product or other OpenSSL toolkit the verb may already be available.

```
$ SHOW SYMBOL OPENSSL
  OPENSSL == "$ SSL1$EXE:OPENSSL"
```

3. Specify a source of lots of "random" data (can be any big file for the purposes of this exercise).

```
$ RANDFILE = "WASD_EXE:HTTPD_SSL.EXE"
```

4. Find the template configuration file. You will need to specify this location in a step described below. Should be something like the following.

```
WASD_ROOT:[SRC.OPENSSL-version.WASD]TEMPLATE.CNF
```

5. Generate your private key (RANDFILE data is used by this). The output from this looks something like what's shown. Notice the pass phrase prompts. **This is your private key, don't forget it!**

```
$ OPENSSL GENRSA -DES3 -OUT SERVER.KEY 1024

Generating RSA private key, 1024 bit long modulus
.....++++++
......++++++
e is 65537 (0x10001)
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

6. Generate the Certificate Signing Request using syntax similar to the following (this is where you are required to specify the location of the configuration template). Note that there are quite a few fields - **GET THEM RIGHT!** They need to be unique and local - they're your distinguishing name (DN). "Common Name" is the host you want the certificate for. It can be a fully qualifier host name (e.g. "klaatu.local.net"), or a local *wildcard* (e.g. "*.local.net") for which you may pay more.

```
$ OPENSSL REQ -NEW -KEY SERVER.KEY -OUT SERVER.CSR -CONFIG -
WASD_ROOT:[SRC.OPENSSL-0_9_6B.WASD]TEMPLATE.CNF
```

```
Using configuration from template.cnf
Enter PEM pass phrase:
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:AU
State or Province Name (full name) [Some-State]:South Australia
Locality Name (eg, city) []:Adelaide
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example
Organizational Unit Name (eg, section) []:WASD
Common Name (eg, YOUR name) []:klaatu.local.net
Email Address []:Mark.Daniel@wasd.vsm.com.au
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

7. That's it! You should have two files in your default directory.

```
SERVER.CSR;1               2  14-MAR-2002 04:38:26.15
SERVER.KEY;1               2  14-MAR-2002 04:31:38.76
```

Keep the SERVER.KEY file secure. You'll need it when you receive the certificate back from the CA.

The SERVER.CSR is what you send to the CA (usually by mail or Web form). It looks something like the following

```
$ TYPE SERVER.CSR
-----BEGIN CERTIFICATE REQUEST-----
MIIBPTCB6AIBADCBhDELMAkGA1UEBhMCWkExFTATBgNVBAgTDFdlc3Rlcm4gQ2Fw
ZTESMBAGA1UEBxMJQ2FwZSBUb3duMRQwEgYDVQQKEwtPcHBvcnR1bml0aTEYMBYG
A1UECxMPT25saW5lIFNlcnZpY2VzMRowGAYDVQQDExF3d3cuZm9yd2FyZC5jby56
YTBaMA0GCSqGSIb3DQEBAQUAA0kAMEYCQQDT5oxxeBWu5WLHD/G4BJ+PobiC9d7S
6pDvAjuyC+dPAnL0d91tXdm2j190D1kgDoSp5ZyGSgwJh2V7diuuPlHDAgEDoAAw
DQYJKoZIhvcNAQEEBQADQQBf8ZHIu4H8ik2vZQngXh8v+iGnAXD1AvUjuDPCWzFu
pReiq7UR8Z0wiJBeaqiuvTDnTFMz6oCq6htdH7/tvKhh
-----END CERTIFICATE REQUEST-----
```

You can see the details of this file using

```
$ OPENSSL RSA -NOOUT -TEXT -IN SERVER.CSR
```

After Receiving The Certificate

Once the signed certificate has been issued by the Certificate Authority it can be placed directly into the server configuration directory, usually WASD_ROOT:[LOCAL], and configured for use from there. Using the certificate direct from the CA requires that the private key password be given to the server each time (Section 4.3.9). It is possible to embed the password into the certificate key so that this is not required.

Remember to keep original files secure, only work on copies!

1. Assign a foreign verb for the OPENSSL application. The location may vary a little depending on which OpenSSL package you have installed.

   ```
   $ OPENSSL == "$WASD_ROOT:[SRC.OPENSSL-version.AXP.EXE.APPS]OPENSSL.EXE"
   ```

   When using the HP SSL1 for OpenVMS product or other OpenSSL toolkit the verb may already be available.

   ```
   $ SHOW SYMBOL OPENSSL
     OPENSSL == "$ SSL1$EXE:OPENSSL"
   ```

2. Go to wherever you want to do the work.

   ```
   $ SET DEFAULT WASD_ROOT:[LOCAL]
   ```

3. You may require these additional steps (based on user experience):

   - VeriSign sent certificate with headers like this:

     ```
     -----BEGIN PKCS #7 SIGNED DATA-----
     -----END PKCS #7 SIGNED DATA-----
     ```

     Using an editor, ensure the header/trailer looks this:

     ```
     -----BEGIN PKCS7-----
     -----END PKCS7-----
     ```

   - Then into the required intermediate format:

     ```
     $ OPENSSL pkcs7 -print_certs -in SERVER.CERT -outform DER -out CERTIFICATE.PEM
     ```

   - A *readable* version of the new file can be viewed using:

     ```
     $ OPENSSL x509 -noout -text -in CERTIFICATE.PEM
     ```

4. Using the original key file embed your password into a copy. When prompted "Enter PEM pass phrase:" enter the password.

   ```
   $ OPENSSL rsa -in SERVER.KEY -out WORK.PEM
   ```

5. Append this password-embedded key file to your certificate file.

   ```
   $ COPY CERTIFICATE.PEM,WORK.PEM CERTIFICATE.PEM;0
   ```

6. Delete the temporary file.

   ```
   $ DELETE WORK.PEM;*
   ```

4.5 SSL CGI Variables

CGI variables specific to SSL transactions optionally may be enabled using WASD_CONFIG_MAP mapping rules. (See “WASD Web Services - Install and Config” document, “Request Processing Configuration” section.) The may be done on a specific per-path or general CGI basis. Two variations are available, one reflecting Purveyor Secure Web Server style variables, the other the Apache *mod_ssl* style. In the following examples, due to length of particular items, some in this example are displayed wrapped. Also, where some ASN.1 records are duplicated (as in SSL_CLIENT_S_DN), some variables will contain newline characters (0x10)

between those elements (e.g. SSL_CLIENT_S_DN_OU). The line breaks in the examples do not necesarily reflect those characters.

set /path/* SSLCGI=apache_mod_ssl

```
WWW_SSL_CIPHER == "AES128-SHA"
WWW_SSL_CIPHER_ALGKEYSIZE == "128"
WWW_SSL_CIPHER_USEKEYSIZE == "128"
WWW_SSL_PROTOCOL == "TLSv1.2"
WWW_SSL_SERVER_A_KEY == "rsaEncryption"
WWW_SSL_SERVER_A_SIG == "sha1WithRSAEncryption"
WWW_SSL_SERVER_I_DN == "/C=AU/ST=SA/L=Adelaide/O=WASD CA Cert
/OU=OpenSSL 1.0.1j Testing Only/CN=WASD VMS Web Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_I_DN_C == "AU"
WWW_SSL_SERVER_I_DN_CN == "WASD VMS Web Services"
WWW_SSL_SERVER_I_DN_EMAIL == "Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_I_DN_L == "Adelaide"
WWW_SSL_SERVER_I_DN_O == "WASD CA Cert"
WWW_SSL_SERVER_I_DN_OU == "OpenSSL 1.0.1j Testing Only"
WWW_SSL_SERVER_I_DN_ST == "SA"
WWW_SSL_SERVER_M_SERIAL == "01"
WWW_SSL_SERVER_M_VERSION == "3"
WWW_SSL_SERVER_S_DN == "/C=AU/ST=SA/L=Adelaide/O=WASD Server Cert
/OU=OpenSSL 1.0.1j Testing Only/CN=WASD VMS Web Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_S_DN_C == "AU"
WWW_SSL_SERVER_S_DN_CN == "WASD VMS Web Services"
WWW_SSL_SERVER_S_DN_EMAIL == "Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_S_DN_L == "Adelaide"
WWW_SSL_SERVER_S_DN_O == "WASD Server Cert"
WWW_SSL_SERVER_S_DN_OU == "OpenSSL 1.0.1j Testing Only"
WWW_SSL_SERVER_S_DN_ST == "SA"
WWW_SSL_SERVER_V_END == "Nov 16 13:02:05 2024 GMT"
WWW_SSL_SERVER_V_START == "Nov 19 13:02:05 2014 GMT"
WWW_SSL_SESSION_ID == "b72812e716f1f20c983935db08ad8ede3af786ffd505b4a2d707eddf8d07dcd9"
WWW_SSL_VERSION_INTERFACE == "HTTPd-WASD/10.4.0 OpenVMS/AXP SSL"
WWW_SSL_VERSION_LIBRARY == "OpenSSL 1.0.1j 15 Oct 2014"
```

The Apache *mod_ssl* client certificate details described in Section 4.3.18 above are not shown in the above example but would be included if the request was X.509 authenticated.

X509 certificate extensions are in general visible from WATCH and accessible via CGI variables when enabled using SET *SSLCGI=apache_mod_ssl_extens* and *SSLCGI=apache_mod_ssl_client* path mappings.

set /path/* SSLCGI=purveyor

```
WWW_SECURITY_STATUS == "SSL"
WWW_SSL_CIPHER == "AES128-SHA"
WWW_SSL_CIPHER_KEYSIZE == "128"
WWW_SSL_CLIENT_AUTHENTICATED == "TRUE"
WWW_SSL_CLIENT_CA == "/O=VeriSign, Inc./OU=VeriSign Trust Network
/OU=www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98
/CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated"
WWW_SSL_CLIENT_DN == "/O=VeriSign, Inc./OU=VeriSign Trust Network
/OU=www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98
/OU=Persona Not Validated/OU=Digital ID Class 1 - Netscape
/CN=Mark Daniel/Email=mark.daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_CA == "/C=AU/ST=SA/L=Adelaide/O=WASD CA Cert
/OU=OpenSSL 1.0.1j Testing Only/CN=WASD VMS Web Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_DN == "/C=AU/ST=SA/L=Adelaide/O=WASD Server Cert
/OU=OpenSSL 1.0.1j Testing Only/CN=WASD VMS Web Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_VERSION == "TLSv1.2"
```

Note that this example also shows *SSL_CLIENT_* . . . variables. These will only be present if the request is X.509 certificate authenticated.

4.6 SSL Service Evaluation

This section is just the barest introduction to a significant topic.

Qualys SSL Lab

"How well do you know SSL? If you want to learn more about the technology that protects the Internet, you've come to the right place."

https://www.ssllabs.com/

Not necessarily an endorsement by WASD but a useful resource in itself.

Provides a *free and unencumbered*, comprehensive SSL Server test service

https://www.ssllabs.com/ssltest/

reporting on certificate status, protocol version, cipher suites, handshakes with various simulated clients, and protocol details including known vulnerabilities. It also summarises the report with a colour-coded rating.

At Home

So to speak.

The OPENSSL command-line application

http://wiki.openssl.org/index.php/Command_Line_Utilities

provides a configurable client for checking and testing various aspects of server configuration and behaviour. The basic operation represented by the command-line

```
$ openssl s_client -host <host name or address> -port 443
```

provides a comprehensive report including certificates and certificate chain, the protocol version and cipher negotiated, along with more esoteric elements of TLS/SSL. Some data have been 8< snipped 8< for brevity in the following example.

```
$ openssl s_client -host klaatu.private -port 443
WARNING: can't open config file: SSLROOT:[000000]openssl.cnf
CONNECTED(00000003)
depth=0 C = AU, ST = SA, L = Adelaide, O = WASD Server Cert, OU 8< snip 8<
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = AU, ST = SA, L = Adelaide, O = WASD Server Cert, OU 8< snip 8<
verify error:num=27:certificate not trusted
verify return:1
depth=0 C = AU, ST = SA, L = Adelaide, O = WASD Server Cert, OU 8< snip 8<
verify error:num=21:unable to verify the first certificate
verify return:1
---
Certificate chain
 0 s:/C=AU/ST=SA/L=Adelaide/O=WASD Server Cert/OU=OpenSSL 1.0.1 8< snip 8<
   i:/C=AU/ST=SA/L=Adelaide/O=WASD CA Cert/OU=OpenSSL 1.0.1j Te 8< snip 8<
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIFsjCCBJqgAwIBAgIBBDANBgkqhkiG9w0BAQQFADCBtjELMAkGA1UEBhMCQVUx
8< snip 8<
pErvrfr69iDbJbhO+mRmIkZIXHc5CFV/M1zzLD5240ixxu/d6nAUBhGba0W4Kste
x1SgLJ0BqFTjegxuHRXkK5lOlY11Hw==
-----END CERTIFICATE-----
subject=/C=AU/ST=SA/L=Adelaide/O=WASD Server Cert/OU=OpenSSL 1. 8< snip 8<
issuer=/C=AU/ST=SA/L=Adelaide/O=WASD CA Cert/OU=OpenSSL 1.0.1j  8< snip 8<
---
No client certificate CA names sent
---
SSL handshake has read 1791 bytes and written 625 bytes
---
New, TLSv1/SSLv3, Cipher is AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : AES256-GCM-SHA384
    Session-ID: 61FEC1629DA3E675AA124223CDB9CB5AB7701D872E85E15 8< snip 8<
    Session-ID-ctx:
    Master-Key: F4260DFE9A7370B3EA85D22D89DB8A7925C655159C3C509 8< snip 8<
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 63 d6 2a 84 19 fe f6 9a-13 60 e1 8a 65 dd f9 fc   c.*......'..e...
8< snip 8<
    00a0 - 9a 2d 29 9b 8e aa ab 69-11 0d 45 ed 63 48 f5 4f   .-)....i..E.cH.O
```

```
    Start Time: 1415828121
    Timeout   : 300 (sec)
    Verify return code: 21 (unable to verify the first certificate)
---
bad select 38
$
```

The “bad select 38” is a VMS (C-RTL) limitation and on another platform the default use of -s_client will prompt for an HTTP request line, send that to the server, and report the response.

Checking whether a specific protocol version is enabled on a site:

```
$ openssl s_client -ssl2 -host <host name or address> -port 443
$ openssl s_client -ssl3 -host <host name or address> -port 443
$ openssl s_client -tls1 -host <host name or address> -port 443
$ openssl s_client -tls1_1 -host <host name or address> -port 443
$ openssl s_client -tls1_2 -host <host name or address> -port 443
```

The following example shows a server test where the protocol version is NOT supported.

```
$ openssl s_client -ssl3 -host klaatu.private -port 443
8< snip 8<
SSL handshake has read 7 bytes and written 0 bytes
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : SSLv3
    Cipher    : 0000
8< snip 8<
```

4.7 SSL References

The following provide a starting-point for investigating SSL and OpenSSL further (verified available at time of publication).

- http://www.openssl.org/
  OpenSSL Project. This site is the prime source for the full toolkit, documentation, related links, news and support via mailing lists, etc.
  http://wiki.openssl.org/
  OpenSSL Wiki

- http://h71000.www7.hp.com/openvms/security.html
  Information regarding HP SSL1 (Secure Sockets Layer) for OpenVMS **may** be found somewhere within this link (though it’s been a bit of a moving target).
  Perhaps here: http://h71000.www7.hp.com/openvms/security.html#ssl
  Perhaps from the OpenVMS (top-level) documentation URL: http://www.hp.com/go/openvms/doc

- http://chimera.labs.oreilly.com/books/1230000000545/ch04.html
  Ilya Grigorik’s - Transport Layer Security (TLS)

  From the excellent http://chimera.labs.oreilly.com/books/1230000000545/index.html

- http://en.wikipedia.org/wiki/Transport_Layer_Security
  Wikipedia - Transport Layer Security (SSL)
- http://en.wikipedia.org/wiki/OpenSSL
  Wikipedia - OpenSSL
- http://en.wikipedia.org/wiki/Public_key_infrastructure
  Wikipedia - Public-Key Infrastructure
- https://istlsfastyet.com/
  https://www.maxcdn.com/blog/ssl-performance-myth/
  Why you should avoid TLS - **NOT!**
- https://www.ssllabs.com/
  Qualys SSL Labs
  https://www.ssllabs.com/ssltest/
  SSL Server Test
- https://www.feistyduck.com/books/openssl-cookbook/
  OpenSSL Cookbook by Ivan Ristic (of Qualys Labs)
  As promoted by OpenSSL.org

Chapter 5
HTTP/2

HTTP/2 is the most recent standard (RFC 7540, 2015) for implementing how HTTP is represented by, and transported between, client and server. It is not a ground-up rewrite of the established standard, HTTP/1.1 (RFC 2616, 1999). Those elements and semantics remain substantially the same. Instead HTTP/2 modifies how the data is encapsulated (framed) and transferred between agents, abstracting the complexity of this within the new protocol layer, leaving the application level largely insulated from change. As a result all existing HTTP/1.1 web-based environments should be able to continue without modification.

The focus of the protocol is on performance, in particular end-user perceived page rendering and web application responsiveness. With the original web use case being a relatively simple, single resource request-response, and early markup involving text with a few illustrative images, the single network connection, back-to-back request-response paradigm was simple to implement and worked well enough. In short time this moved to multiple network connections, each loading elements in parallel as the complexity and density of the individual elements on the pages increased, and to the introduction of HTTP/1.1 *pipelining* (back-to-back requests over a single connection) in an attempt to avoid request-response-request latency. Modern web documents and applications tend to have dozens of fine-grained elements that dynamically load resources based on the content of the page and/or user interaction. The single, then multiple network connections, each with its round-trip TCP connection establishment overhead and request-response blocking of resources, did not scale effectively. HTTP/2 replaces it with a single TCP connection on which multiple resources concurrently can be requested, pushed, and transferred. A more rigorous and effective implementation of the pipeline concept.

While multiplexing communication over a single network connection is a core performance technology there are other contributing elements. The framing layer uses binary tokens and parameters. The plain-text request and response headers of HTTP/1.*n* are replaced with tokenised, encoded and dynamically cached equivalents, commonly providing compression in excess of eighty percent. The relationship and priority of resources can be established allowing inferior resources to be delivered after or dependent on superior ones. The HTTP/2 server can send multiple responses to a single request. Known as *server push* it can be used to pre-load the browser (cache) with resouces it has not encountered yet.

HTTP/2 has the potential to place additional load on the client and server in comparison to HTTP/1.*n*. One particular consideration for WASD sites is the *stream concurrency* setting of the HTTP/2 connection. The server specifies to the client the maximum number of concurent request-response (and server push) *streams* it will accept. RFC 7540 contains, "This limit is directional: it applies to the number of streams that the sender permits the receiver to create. Initially, there is no limit to this value. It is recommended that this value be no smaller than 100, so as to not unnecessarily limit parallelism." This translates to a hypothetical ten browsers connected to the site each with up to one hundred concurrent streams, or potentially one thousand active requests! Time to check those server configuration and SYSGEN parameters . . .

Note that HTTP/1.1 has recently been revisited with RFC 7230 family of specifications (2014) providing some clarifications and refinements on the original.

5.1 WASD HTTP/2

WASD HTTP/2 implements all of the essential requirements of RFC 7540 (naturally enough). This includes the framing protocol, datagram (message) and stream management, header compression (RFC 7541), connection settings and flow control, along with HTTP/2 connection establishment and termination (TLS ALPN and HTTP upgrade). It does not ((perhaps) currently) provide server-push or stream prioritisation and dependency.

Prior to the introduction of HTTP/2, WASD's fundamental abstraction was the request, with each request interfacing directly with the network stack. With an HTTP/2 protocol connection somewhat supplanting the role of a Transmission Control Protocol (TCP) connection in HTTP/1.*n*, a new level of communication abstraction was required between the request processing and the network processing. It should be noted that HTTP/2 itself is transported on TCP.

Another new layer of abstraction required interfacing each protocol's request/response header formats with the underlying server processing (avoiding excessive duplication of code). HTTP/1.*n* has a plain-text, carriage-control separated format, while HTTP/2 has a binary, compressed, lookup-table oriented format (RFC 7541). The layer was implemented using a *key-value* dictionary.

The accomodations for handling both HTTP/2 and HTTP/1.1, along with related and ancilliary design and code changes, have not measurably impacted overall WASD performance, although as noted below there is a server process CPU impost associated with HTTP/2.

HTTP/2 and WATCH

WATCH reports have a new network item: [x]HTTP/2. This provides a detailed overview of the underlying framing and connection management exchanges between client and server. WATCH reports are available to HTTP/2 connected clients with one consideration. Due to multiplexed requests over the single network connection, WATCHing the [x]HTTP/2 item of another request in the same browser (using the same HTTP/2 connection - and there *can* be multiple from a single browser) is not possible (or at least more code than it's worth). The HTTP/2 activity of the WATCHing generates more report items which generate . . . a descent into reporting oblivion.

WASD detects when a request is initiated on the same HTTP/2 connection as an [x]HTTP/2 WATCHing client and advises

```
|Time_______|Module__|Line|Item|Category__|Event...|
|22:00:55.22 WATCH    1823 0004 CONNECT    HTTP/2 with 192.168.1.2,62446 on https://klaatu.
|22:00:55.22 WATCH    1454 0004 CONNECT    HTTP/2 rabbit hole|
```

Such a request is not reported on further.

Workarounds?

- WATCH from an independent browser instance. Often requires a separate host or different browser (e.g. Chrome and Firefox on the same host).
- Have an HTTP/1.1 (only) service on the same server and use WATCH from that.

5.2 HTTP/2 and Performance

With HTTP/2 not modifying the fundamentals of HTTP/1.1 semantics the commonly touted payoff for all the additional complexity (in implementation) is performance. While this is often stated in terms of page rendering speeds or web application responsiveness there is another significant measure of performance - efficiency. HTTP/2 much more efficiently utilises each network (TCP) connection, as well as reducing the (time and processing) overhead of setting-up and tearing-down of each of these required for parallelism under HTTP/1.1.

Is it all worth it? As might be expected - that depends.

There are a number of sufficiently good analyses of both the factors that affect HTTP/2 performance and the actual performance relative to HTTP/1.1. See the references section and search the Web. This section contains some observations made during WASD HTTP/2 development. All of these seem to correspond with others' observations, as well as what might reasonably be expected considering the strategies employed by the protocol.

- For simple request-response use cases (e.g. download a file) HTTP/2 makes no observable performance difference.
- Where multiple resources need to be loaded by a page the measurable performance improvement is proportional to the number of resources and the latency of the network.
- In a low-latency environment such as the average LAN (e.g. 5mS RTT) HTTP/2 makes minimal difference irrespective of the number of resources loaded (until it reaches rediculous quantities).
- In a high-latency environment such as a VPN spanning half the globe (e.g. 350mS RTT) HTTP/2 makes an obvious and of course measurable improvement for anything other than a trivial number of resources.
- On a CPU constrained system HTTP/1.*n* is significantly more responsive than HTTP/2. This unsurprising considering the explicit multiplexing and header marshalling employed by HTTP/2.

- On the developer's bench there is ~10% more CPU consumed for the same load profile** via HTTP/2 compared to HTTP/1.1 for similar durations. This is (probably) due to header compression and multiplexed stream processing. It is (probably) offset (to some degree) by fewer resources consumed in the network stack managing the multiple TCP connections of HTTP/1.1.

  As also related in Chapter 11, using the same load profile as above** and using HTTP/1.1, WASD v11.0 compared to v10.4 showed ~5% additional CPU and duration. This is (probably) largely due to dictionary processing.

  ** *100 individual files, size 2kB to 250kB, 50 concurrent, ~30% CPU utilisation (~5% USER mode, mostly INTERRUPT servicing), batched 10,000 at a time over a LAN.*

- After some months accessing WASD HTTP/2 over various LANs and WANs the developer, FWIW, can't shake the perception that it *seems* generally more responsive.

  YMMV!

Performance Assessment

The simplest tool for getting a *feel* for, and elementary measurement of HTTP/2 may be found in the WASD_ROOT:[EXERCISE] directory. The document DOTTY.HTML and its companion files provide a page that loads a selectable number of resources (images) in a consistent and reproducible manner. This DOTTY.HTML can be accessed via unencrypted HTTP (http://), encrypted HTTP (https://) and services configured to provide HTTP/2 or HTTP/1.1. Using these combinations with the selectable volume of resources, elementary comparisons may be made in target environments.

The Server Admin, HTTP Report (Chapter 9) contains comparative duration and bytes-per-second minimum/maximum/average for total server HTTP/2 and HTTP/1.*n* requests. These cannot simply be taken at face value without some consideration of the respective load profile but under controlled conditions can provide useful metrics.

Other development and load/performance tools were employed from a Linux platform. For someone educated in computing during the (19)70s, the availability of VM technology for such purposes is just brilliant! "*But you know, we were happy in those days, though we were poor.*"

Indispensible were:

https://nghttp2.org/documentation/nghttp.1.html
https://nghttp2.org/documentation/h2load.1.html

Many thanks to the developer(s) of this package.

5.3 HTTP/2 Configuration

While effectively transparent to the end-user, HTTP/2 has some aspects that need to be carefully considered by the server administrator.

- The level of (request) concurrency suggested by RFC 7540 section 6.5.2 would likely require redimensioning a web server and possibly the supporting system. Environments historically expecting per-client resource demand to be limited by the number of concurrent (HTTP/1.*n*) network connections an agent will deploy per origin server, often limited to less than a dozen, might behave entirely differently when presented with many dozens,

or potentially hundreds of requests. WASD's default of 100 is the RFC recommendation in part because browsers tend to open multiple connections to maintain the parallelism sought, so a reduction in HTTP/2 stream concurrency often just increases HTTP/2 connection concurrency.

- Secure HTTP requires a minimum of TLS 1.2 with SNI and ALPN (RFC 7540 section 9.2).

- The ciphers available for use with HTTP/2 secure HTTP are quite specific (at least in what the RFC prohibits - RFC 7540 Appendix A). This and the overall encryption requirements for HTTP/2 can cause issues with established (older) agents and with mainstream browsers strictly enforcing the RFC definitions making support for combined /2-/1.1 services sometimes problematic.

  Use of elliptic curve ciphers (ECDHE), as an element of Perfect Forward Security (PFS), is mandated for HTTP/2 (RFC 7540 section 9.2.2). The keys for the elliptic curve ciphers are stored in PEM-encoded files located in WASD_ROOT:[LOCAL]. These can be copied from the WASD OpenSSL package using

  ```
  $ copy WASD_ROOT:[SRC.OPENSSL-n_n_n.WASD.CERT]DH_PARAM_*.PEM WASD_ROOT:[LOCAL]
  ```

  or locally generated as described in Section 4.3.5.

  This SSL configuration and minimum cipher list seems to work for all major browsers at the time of writing:

  ```
  # WASD_CONFIG_GLOBAL
  [SecureSocket]  enabled
  [SSLversion]  TLSvall
  [SSLoptions]  +OP_CIPHER_SERVER_PREFERENCE
  [SSLcipherList] EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH:-DSS:
  ```

  YMMV!

- TLS renegotiation (e.g. for a client certificate) must not be performed on an HTTP/2 secure connection. This precludes having selected paths perform authorisation based on X509 and means that the service itself must request a client certificate at connection establishment (RFC 7540 section 9.2.1).

- While the protocol provides for HTTP/2 using non-TLS (non-SSL) connections the major browsers (Chrome, Edge (MSIE), FireFox, Safari) only support it when using TLS. To *encourage* naive users to a TLS service the following mapping rule approach may be used to redirect non-TLS home page connections.

  ```
  # WASD_CONFIG_MAP
  [[*:80]]
  if (!ssl:) redirect / https:///
  ```

5.3.1 Global Configuration

HTTP/2 and its features are globally enabled and configured using directives contained in the WASD_CONFIG_GLOBAL configuration file.

HTTP/2 Global Configuration

Directive	Description	Default
[Http2Protocol]	enabled or disabled on a whole-of-server basis	disabled
[Http2FrameSizeMax]	maximum frame size in octets (bytes) the server is prepared to receive	16384
[Http2HeaderListMax]	maximum number of octets (bytes) permitted in a received header once uncompressed	65535
[Http2HeaderTableMax]	maximum number of bytes permitted in the server-end header cache	4096
[Http2PingSeconds]	number of seconds between connection RTT pings	300
[Http2StreamsMax]	maximum number of concurrent streams (requests) the server permits on the connection	32
[Http2InitWindowSize]	initial window size (number of octets in transit) for flow-control purposes	6291456

These largely reflect settings and defaults from RFC 7540 6.5.1

- The minimum frame size is defined by the RFC at 16384.
- WASD automatically pings a connection every configured seconds. The latest value is available as real-number milliseconds in dictionary entry “http2_ping” and CGI variable HTTP2_PING.

5.3.2 Service Configuration

Using the WASD_CONFIG_SERVICE directive [ServiceHttp2Protocol] HTTP/2 may be disabled on a per-service basis. The default is enabled if HTTP/2 is enabled globally.

5.3.3 Mapping Set Rules

WASD request processing rules may be used on a per-path basis to modify (some) global configuration settings and provide other WevDAV configuation. See “WASD Web Services - Install and Config” .

HTTP/2 Set Rules

Rule	Description
HTTP2=PROTOCOL=1.1	send a “HTTP_1_1_REQUIRED” error causing the client to use HTTP/1.1 (RFC 7540 section 7)
HTTP2=SEND=GOAWAY	send a “GOAWAY” frame to the client resulting in it dropping the HTTP/2 connection
HTTP2=SEND=PING	send a “PING” frame to the client calculating the Round Trip Time (RTT) of the connection
HTTP2=SEND=RESET	send a “RST_STREAM” frame to the client causing it to drop the HTTP/2 stream (request in progress)
HTTP2=STREAMS=MAX=*integer*	set the maximum concurrent streams on a per-path basis
HTTP2=WRITE=*low* | *normal* | *high*	When request data is written it is queued at the specified priority, where high priority are written before normal (default) and low priority, and normal priority before low. This is only for associated stream (request) and is not a connection or whole-of-server prioritisation.

Use path SETings to prioritise some resources (e.g. CSS and JavaScript) over others (e.g. images) and potentially improve page rendering speed. Where multiple concurrent requests are being serviced on the one HTTP/2 connection this will deliver the *high*er priority content before others.

```
# WASD_CONFIG_MAP
SET **.css http2=write=high
SET **.js  http2=write=high
```

5.4 HTTP/2 Detection

A request using HTTP/2 may be detected during processing with the *http2:* conditional.

```
if (http2:)
   do this
endif
```

See “WASD Web Services - Install and Config” .

A script may detect HTTP/2 using the REQUEST_PROTOCOL CGI variable with the value “HTTP/2”. Other protocol versions are similarly represented.

A Server-Side Includes (SSI) document can use variations on the following construct (and similar to the script suggestion immediately above) to detect and process the request protocol.

```
<!--#if var={request_protocol} eqs="HTTP/2" -->
HTTP/2
<!--#else-->
HTTP/1.n
<!--#endif-->
```

This is demonstrated in the example SSI document: WASD_ROOT:[EXERCISE]SHTML.SHTML

At the time of writing there is no browser-supported mechanism for a dynamic document (i.e. JavaScript) determining the underlying HTTP protocol used to access a resource. To access this information the server must be used. The suggested method, and the one employed by the DOTTY.HTML tool described above, is to provide one JavaScript source for HTTP/2 and another for everything else.

The document would contain

```
<script type="text/javascript" src="/example-path/http.js"></script>
```

and the server configuration

```
# WASD_CONFIG_MAP
if (http2:)
   map /example-path/http.js /example-path/http2.js
else
   map /example-path/http.js /example-path/http1.js
endif
```

where each contains a minimum variable setting or similar flag detectable by the document.

5.5 HTTP/2 References

The following provide a starting-point for investigating HTTP/2 (verified available at time of publication).

- https://http2.github.io/
  Home page for HTTP/2 maintained by the IETF HTTP Working Group.
- https://en.wikipedia.org/wiki/HTTP/2
- https://httpwg.github.io/specs/rfc7540.html
  https://tools.ietf.org/html/rfc7540
  HTTP/2 specification
- https://httpwg.github.io/specs/rfc7541.html
  https://tools.ietf.org/html/rfc7541
  HPACK (header compression) specification
- https://httpwg.github.io/specs/rfc7230.html
  https://tools.ietf.org/html/rfc7230
  Most recent HTTP/1.1 specifications (30, 31, 32, 33, 34 and 35)
- http://http2-explained.haxx.se/
  Useful overview of HTTP/2 by the developer of cURL.

- http://chimera.labs.oreilly.com/books/1230000000545/ch12.html
  Another useful and more detailed overview of the protocol.
- http://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html
  A concise and useful summary.
- https://insouciant.org/tech/http-slash-2-considerations-and-tradeoffs/
  An interesting read, with particular emphasis on network performance and behaviour.
- https://blog.cloudflare.com/tools-for-debugging-testing-and-using-http-2/
  Not much here for VMS but a useful survey nonetheless.

Chapter 6
WebDAV

Web-based Distributed Authoring and (not) Versioning for the WASD package.

Effective WASD WebDAV file-space (without significant naming constraints) relies on being hosted on ODS-5 volumes. Behaviour hosting file-space on ODS-2 volumes is untested (though possible provided file naming is constrained to ODS-2 conventions).

WASD WebDAV methods and request headers, etc., are also propagated to the scripting environment and so functionality may be implemented using CGI, CGIplus or RTE based applications.

WASD proxy-serving supports WebDAV methods, header fields, etc.

Generally WebDAV clients are applications other than browsers and so response bodies with human-readable error explanations are unnecessary and consume bandwidth to no good purpose, and so not provided.

File-systems are notoriously latent components relative to the rest of the system (more so with VMS). Any operation to collections (directories) are not going to be atomic and for large collections requiring many sub-operations the potential for the process to be interrupted or otherwise disturbed are enormous. File-systems are not databases amenable to extensive ACID operations.

In addition each file under WebDAV management has the potential for an associated but independent metadata file. This of course means for every DAV-specific resource file activity there is at least a file-system action to check for a metadata file and for some actions such as COPY the potential for an associated but entirely independent file operation.

Of course WebDAV was not intended or designed as a general file-system protocol but one for distributed management of somewhat restricted collections of Web-related resources and so in context probably works well enough.

See sections below on file-system operation method restrictions.

Caution

If using WebDAV in any serious fashion the likes of

```
$ HTTPD/DO=RESTART=NOW
```

during server WebDav file-system modifications is a recipe for inconsistency and/or corruption!

References

These are the resources used during WASD WebDAV development.

- WebDAV in general:
  http://webdav.org/
  http://en.wikipedia.org/wiki/Webdav
  http://tools.ietf.org/html/rfc4918
  http://tools.ietf.org/html/rfc4331 (quota)
  http://tools.ietf.org/html/rfc2518 (obsoleted by RFC 4918)
- WebDAV: Next-Generation Collaborative Web Authoring
  Lisa Dusseault, 2003 ISBN: 0130652083
- Using Expat by Clark Cooper:
  http://en.wikipedia.org/wiki/Expat_(XML)
  http://www.xml.com/pub/a/1999/09/expat/index.html
  http://www.xml.com/lpt/a/47

Client Tools

All these have been used during WASD WebDAV development.

- A comprehensive but not exhaustive list
  http://www.webdavsystem.com/server/access/
  http://www.webdavsystem.com/server/access/clients_comparison
- DAVExplorer - a Java-based GUI Explorer-style file navigation tool
  http://www.davexplorer.org/
- cadaver - a command-line WebDAV client for *x
  http://www.webdav.org/cadaver/
- davfs2 - a mountable WebDAV file-system for Linux
  http://savannah.nongnu.org/projects/davfs2
- The WebDAV URL handling of KDE 4.2 Dolphin (v1.2)
  http://www.webdavsystem.com/server/access/konqueror (yup, I know!)
  In contrast to Gnome as reported below, KDE and its KIO/Dolphin behave extrordinarily well.
- The WebDAV URL handling of Gnome Nautilus (2.26.2, gvfs/1.2.2)
  http://www.webdavsystem.com/server/access/gnome_nautilus
  As at publication, **Gnome/gvfs/Nautilus has quite a number of behavioural problems** with associated Bugzilla items. Don't expect it to behave reasonably!
- The WebDAV handling of Apple Mac OS X (10.6) Finder
  http://www.apple.com/macosx/what-is-macosx/dock-and-finder.html
  http://www.webdavsystem.com/server/access/macosx

- Windows Explorer - and the associated mini-director, et.al., on XP (not Vista).
  See Section 6.6 below.
- Another Windows option - try before you buy (i.e. commercial product).
  "WebDrive is more than just an FTP Client." Indeed! It's functional WebDAV drive-letter client.
  http://www.webdrive.com/
- **And if you really need effective WebDAV on a Windows platform ...**
  "BitKinex integrates the fuctionality of an innovative FTP, SFTP and WebDAV client for Windows."
  And it's FREEWARE!
  http://www.bitkinex.com/

6.1 HTTP Methods Supported

A list of WebDAV methods, what WASD does with them, and any limitations or restrictions. Some of these are familiar HTTP/1.*n* methods and other are RFC 4981 specific. Some of the HTTP/1.*n* methods are overloaded with additional or variant behaviours when used in a WebDAV context. Issues of atomicity with the manipulation of file-system trees containing numbers of individual files makes strict RFC 4918 compliance difficult. See " . . . Restrictions" below.

WebDAV HTTP Methods

Method	Description
COPY**	Reproduces both single resources (files) and collections (directory trees). Will overwrite files (if specified by the request) but will respond 209 (Conflict) if it would overwrite a tree.
DELETE**	deletes files and directory trees
GET	just the vanilla HTTP/1.1 behaviour
HEAD	ditto
LOCK**	see WEBDAV LOCKING below
MKCOL**	create a directory
MOVE**	Moves (rename or copy) a file or a directory tree. Will 'overwrite' files (if specified by the request) but will respond 209 (Conflict) if it would overwrite a tree.
OPTIONS	If WebDAV is enabled and available for the path this reports the WebDAV extension methods
PROPFIND**	Retrieves the requested file characteristics, DAV lock status and 'dead' properties for individual files, a directory and its child files, or a directory tree.
PROPPATCH**	set and remove 'dead' meta-data properties

Method	Description
PUT	Against a WebDAV resource behaves a little differently to historical WASD implementation of PUT.
UNLOCK**	see WebDAV locking below
	***WebDAV RFC 4918 method*

WASD Statistics Reports gather WebDAV related data. Where a method can be used both for vanilla HTTP/1.*n* and WebDAV purposes it is counted in WebDAV statistics if the request header contains some other indication of a WebDAV activity.

6.1.1 COPY Restrictions

Does not comply with the overwrite:T directive for collections (does so for files). Will not preemptively delete the existing tree. It returns a 209 (Conflict) response instead.

COPY does not maintain collection consistent URL namespace if a member resource cannot be moved as required by RFC 4918. It should maintain the source subtree completely uncopied. Instead it is best-effort and continues copying resources until exhausted. This is consistent with file-system behaviour. The RFC 4918 requirement, while not impossible, is fraught with issues inside a file-system.

6.1.2 DELETE Restrictions

Deletion of collections is particularly fraught with issues for a file-system. In userland it is almost impossible to predetermine if an individual file in a directory tree is going to resist deletion (due to locking, protections, etc) and in kernel land it's probably no easier. It leaves the undeleted tree hierachy (resource ancestors) intact. This is RFC 4918 compliant however!

So, in the case of WASD WebDAV it's just best-effort and if something down the tree won't disappear, it just reports the failure in the 207 response and carries merrily on through the tree regardless. This IS acceptable WebDAV server behaviour!

6.1.3 MOVE Restrictions

Does not comply with the overwrite:T directive for collections (does so for files). Will not currently pre-emptively delete the existing tree. It returns a 209 (Conflict) response instead.

MOVE first attempts to rename the file or directory. This is reasonably efficient, especially for directory trees but obviously only suitable for a target on the same disk volume. If a rename failure is due to a different device it falls back to using a COPY then DELETE in two separate phases. Needless-to-say this is hardly atomic and can lead to inconsistencies between source and target.

MOVE does not maintain collection consistent URL namespace if a member resource cannot be moved as required by RFC 4918. It should maintain the source subtree unmoved. Instead it is best-effort and continues moving resources until exhausted. This is consistent with file-system behaviour. The RFC 4918 requirement, while not impossible, is fraught with issues inside a file-system.

6.1.4 If: Restrictions

The conditional "If:" request header field does not have full RFC 4918 support. It implements lock token and etag token processing with parenthetical OR and NOT processing. For unsupported features WATCH reports that the header was not understood and always returns an abort status. WebDAV "If:" processing is an extrodinarily complex kludge for on-the-fly decision making by the server and much of what I have read indicates most clients only ever use extremely simple conditions anyway.

6.2 WebDAV Configuration

WebDAV and its features are globally enabled and configured using directives contained in the WASD_CONFIG_GLOBAL configuration file.

WebDAV Global Configuration

Directive	Description
[PutMaxKBytes]	maximum size of a file (PUT and POST)
[WebDAV]	This directive enables and disables WebDAV.
[WebDAVlocking]	Enables and disables WebDAV locking.
[WebDAVlockTimeoutDefault]	see Locking Timeout
[WebDAVlockTimeoutMax]	see Locking Timeout
[WebDAVlockCollectionDepth]	See Locking Depth
[WebDAVmetaDir]	See Section 6.3
[WebDAVquota]	Enables and disables RFC 4331 functionality (disk quota reporting).

In addition these and other configurations are provided on a per-path basis using mapping rules.

6.2.1 WebDAV Set Rules

WASD Request processing rules (see “WASD Web Services - Install and Config”) may be used on a per-path basis to modify (some) global configuration settings and provide other WevDAV configuation.

WebDAV Set Rules

Rule	Description
ODS=NAME=*8BIT* | *UTF8* | *DEFAULT*	When a file is PUT using WebDAV (or upload), for non-7bit ASCII file names use native ODS-5 8bit syntax (default) or UTF-8 encoded character sequences (see Section 6.2.2)
PUT=MAX=<integer> | *	Maximum number of kilobytes file size, if "*" then effectively unlimited (per-path equivalent of the global directive [PutMaxKBytes]).
WEBDAV=[NO]HIDDEN	list (default) or hide U*x *hidden* files (i.e. those with names beginning with period)
WEBDAV=[NO]LOCK	allow/apply WebDAV locking to this path
WEBDAV=[NO]PROFILE	WebDAV access according to SYSUAF profile
WEBDAV=[NO]PROP	allow/apply WebDAV 'dead' property(ies) to this path
WEBDAV=[NO]PUT=LOCK	a resource must be locked before a PUT is allowed
WEBDAV=[NO]READ	WebDAV methods allowed read this tree
WEBDAV=[NO]SERVER	WebDAV access as server account (best effort)
WEBDAV=[NO]WINPROP	when NOWINPROP windows properties are ignored and emulated
WEBDAV=[NO]WRITE	WebDAV methods allowed write to this path (implied read)
WEBDAV=LOCK=TIMEOUT=DEFAULT=	hh:mm:ss
WEBDAV=LOCK=TIMEOUT=MAX=	hh:mm:ss
WEBDAV=META=DIR=	per-path equivalent of global [WevbDAVmetaDir] (see Section 6.3)

An essential function of the path setting rules is for specifying which paths in server Web-space are allowed to be accessed using the WebDAV protocol and what sort of access (read, write, etc.) that path is allowed.

6.2.2 File Naming

By default files that are PUT via WebDAV (or upload) support the ISO Latin-1 character set. ASCII and non-7-bit file names use the native ODS-5 syntax. Where character sets other than ISO Latin-1, or where compatibility with other WebDAV implementations is desired (e.g. Apache), a path can be set to allow file names supplied using UTF-8 sequences.

For example, the English language word “naïve”, , having a diaeresis mark over the “i” character (indicating it is pronounced separately from the preceding vowel) is commonly respresented using the 8 bit character 0xEF, or as the two byte UTF-8 sequence 0xC3AF. This word if used as the file name with a type (extension) of “.TXT” by default would have the sequence of 8-bit characters

```
0x6E 0x61 0xEF 0x76 0x65 0x2e 0x54 0x58 0x54
```

and if the path had been set *ods=name=utf8* the sequence would be

```
0x6E 0x61 0xC3 0xAF 0x76 0x65 0x2E 0x54 0x58 0x54
```

"Index of" (directory) listings will honour a path set *ods=name=utf8* and make the listing character set UTF-8 resulting in a browser correctly rendering the name (WebDAV listings are by definition UTF-8).

File Name Ambiguity

While files and directories created via WebDAV will have a consistent naming schema applied those created by applications or manual operation on the VMS system can result in files that are not accessible with WebDAV.

For example the file name

```
This^_is^_an^_EXAMPLE^.txt.;1
```

would be presented to the client as

```
This is an EXAMPLE.txt
```

which when provided in a URL as

```
This%20is%20an%20EXAMPLE.txt
```

and translated from that URL into the file specification

```
This^_is^_an^_EXAMPLE.txt;1
```

of course will not be able to be accessed.

In addition, the two files

```
This^_is^_an^_EXAMPLE.txt;1
This^_is^_an^_EXAMPLE^.txt.;1
```

are distinct in the file-system, independently parsed from the directory structure, would be presented to the client as consecutive entries having the same name, with only the accessible file name actually available.

```
This is an EXAMPLE.txt
This is an EXAMPLE.txt
```

To avoid this situation a potentially ambiguous file name containing an escaped period and no type (extension) is ignored by directory listings and WebDAV property lists. When an ambiguous file name is detected it is reported in WATCH reports.

Avoid "Interesting" File Names

While most of these are corner-cases it is best to try and avoid *interesting* file names that can challenge the rather convoluted VMS file-system environment. Inaccessible file names cannot of course be deleted or renamed via WebDAV and may result in directory (folder) deletion problems. These situations generally require manual intervention.

6.2.3 File-system Access

Is controlled using the mapping rules:

File-system Access

Rule	Description
WEBDAV=PROFILE	access using request SYSUAF-authenticated security profile
WEBDAV=WRITE	unconditional permission to read/write
WEBDAV=READ	unconditional permission to read
WEBDAV=SERVER	access using server account permissions

All access by WebDAV operations **must have at least one set** against the path. If access is permitted by one of the above settings SYSPRV is enabled to allow that access using the server account. Therefore files and directories should have a SYSTEM:READ+WRITE+EXECUTE+DELETE protection or equivalent ACL permissions, or the access may fail totally or in some part of a supposedly atomic action.

These file-system access settings are applied in the order listed above. That is, if a path successively has one or more of the above settings applied during rule processing, when it comes to applying those access controls, SYSUAF profile is applied, then if no profile SETing access to read/write, then to read-only, then access via the server account.

In addition WebDAV access requires an authorisation rule against each path.

6.2.4 File-system Authorisation

All access by WebDAV operations **must have one set** against the path.

All WebDAV access is a combination of WASD_CONFIG_MAP path setting and WASD_CONFIG_AUTH authorisation permissions. The least permissive of the two overrides the more. The combination of an authorisation rule and a path mapping rule mitigates the chance of opening unintended access into the file-system.

These is the test-bench environment used during development:

```
# WASD_CONFIG_MAP
pass  /dweb/*  /dweb/*  ods=5 webdav=write webdav=nowinprop

# WASD_CONFIG_AUTH
["KLAATU"=WASD_VMS_RW=id]
/dweb/*  r+w
```

Note that WebDAV read/write access is a combination of the mapping and the authorisation rule (mapping WEBDAV=READ overrides authorisation read+write). Expect complications with Microsoft environments.

For test-benching you could avoid authorisation issues completely with:

```
# WASD_CONFIG_AUTH
[world]
/dweb/*  r+w
```

6.2.5 Concurrent Authorisation

A common requirement is to provide concurrent general access and authorised WebDAV acccess to the same Web-space. This is accomplished by using two paths mapped into the same file-system space, the general access (non-authorised) path, and a WebDAV (authorised) path. The WebDAV client uses the authorised path and can then apply WebDAV methods to maintain the resources.

```
# WASD_CONFIG_MAP
pass  /web/*    /web/*  ods=5
pass  /davweb/* /web/*  ods=5 webdav=profile webdav=nowinprop

# WASD_CONFIG_AUTH
["KLAATU"=WASD_VMS_RW=id]
/davweb/*  r+w
```

6.2.6 Real-World Example

The following configuration is taken from a site using WebDAV to allow users to manage their Web presence. The user mapping is a fairly standard configuration for VMS accounts (see "WASD Web Services - Install and Config"). User Web areas are in the [.WWW] subdirectory of the account home area.

```
# WASD_CONFIG_MAP
# general and WebDAV access (order is important)

user  /~*/dav/* /*/www/*  webdav=profile notepad=webdav
user  /~*/dav   /*/www    webdav=profile notepad=webdav
if (pass:-1 && notepad:webdav)  pass  /~*/dav/*  /d1/*/www/*
if (pass:-1 && notepad:webdav)  pass  /~*/dav/*  /d2/*/www/*

user /~*/* /*/www/* dir=access
if (pass:-1)  pass  /~*/*  /d1/*/www/*
if (pass:-1)  pass  /~*/*  /d2/*/www/*
```

The four WebDAV access rules are located before the three general user access rules. The WebDAV rules are more specific. The first USER rule maps subdirectories - and the parent if a trailing slash is included. The second USER rule maps the parent directory for user agents that do not include trailing slash on their directory specifications (most it seems).

The second pair of rules *reverse-maps* the VMS file-system specifications represented by the *result* (right side) of the PASS rule into the path represented by the *template* (left side) of the PASS rule. Mapping from file-specifications to paths is necessary because of the way the PROPFIND method searches the file-system and then reports its results to the client as URLs.

The use of the *notepad* rule with a string of "webdav" (the actual string is not significant as long as it is unique within the rules) is used to conditionally process the reverse-mapping rules. They will be applied only to the requests originally mapped by the USER rules. The *pass:-1* ensures the rules are only applied during reverse-mapping, not during request mapping.

The fifth rules maps general Web access to the user area. Remember, web access is to a user home subdirectory [.WWW].

The sixth and seventh rules *reverse-map* the VMS file-system specifications for the general USER rules for similar reasons to those described above. Why two? The user directories occur across two disk volumes and so each must be reverse-mapped.

```
# WASD_CONFIG_AUTH

["VMS username/password"=WASD_VMS_RW=id]
/~*/dav/*  read+write,profile,https:
/~*/dav    read+write,profile,https:
```

As noted above, WASD WebDAV requires both mapping and authorization rules (even for "world" - or non-authenticated - access).

In this case authorisation is only required for WebDAV access. There are two rules. The first authorises subdirectories and parent directories for agents that supply a trailing slash. The second for agents that do not provide a trailing slash.

Why use . . .

. . . two rules for each location? Why

```
user  /~*/dav/*  /*/www/*
user  /~*/dav    /*/www
```

rather than

```
user  /~*/dav*    /*/www*
```

which would accomplish a *similar* result?

For finer control. The first only matches requests with a path of "/~user/dav/subdir/" and "/~user/dav", whereas the latter matches "/~user/dav/subdir/" and "/~user/dav" and "/~user/david/" and "/~user/davros", etc.

6.3 WebDAV Metadata

Metadata is data (information) about data. WebDAV uses the concept of a resource *property*. There are "live" properties and "dead" properties. Essentially the live properties are the dynamic characteristics of a file-system object represented by creation and modification date-times, object size, etc. WebDAV dead properties are those supplied by WebDAV clients as XML entities and stored associated with the particular WebDAV object, in WASD's case the file-system object (file or directory). WASD also uses the file metadata to store resource lock data (see Section 6.4).

Metadata Files

WASD manages resource metadata using a separate file associated by name with the data file. This is done for reasons of programmatic simplicity and for the convenience of any command-line owner or sysadmin of the resources. No specialised tools are required. This metadata file can be stored in one of three locations.

1. By default, WASD uses a metadata file in the same directory and the same name with "__wasdav" appended to the extension (type). All non-WebDAV WASD functionality

ignores "*.*__wasdav;" files (e.g. directory listing, file GET). Of course other applications (e.g. directory listing) do not.

```
$ DIRECTORY/SIZE/DATE 01234*.*

Directory WEB:[DAVweb]

01234^.56789.TXT;1    0.50KB   8-JUN-2009 23:07:19.26
01234^.56789.txt__wasdav;1
                         1KB  19-JUN-2009 03:20:34.50
0123456789.TXT;1      0.50KB   8-JUN-2009 23:06:59.16
0123456789.txt__wasdav;1
                         1KB  19-JUN-2009 03:19:14.67
```

2. An alternate but still *local* location, is in the WASD_CONFIG_GLOBAL [WebDAVmetadir] globally specified, or per-path *SET /path webdav=meta=dir* directives. If specified as a subdirectory the metadata file is stored in a subdirectory of the data file directory using the same name with "__wasdav" appended to the extension (type). This is owned by the owner of the parent directory. The metadata directory does not appear in WASD WebDAV or file system listings.

   For example, with the global directive

   ```
   # WASD_CONFIG_GLOBAL
   [WebDAVmetaDir] [.^.dav]
   ```

   specifying a subdirectory with a name containing a leading period (i.e. a U*x *hidden* file), the data files

   ```
   Directory WEB:[DAVweb]

   01234^.56789.TXT;1    0.50KB   8-JUN-2009 23:07:19.26
   0123456789.TXT;1      0.50KB   8-JUN-2009 23:06:59.16
   ```

   would have the associated metadata files

   ```
   Directory WEB:[DAVweb.^.dav]

   01234^.56789.txt__wasdav;1
                            1KB  19-JUN-2009 03:20:34.50
   0123456789.txt__wasdav;1
                            1KB  19-JUN-2009 03:20:24.77
   ```

3. The final alternative uses the same directives as above but specifies a full directory path. In this case WebDAV metadata is stored completely separately from the data. This can be anywhere in available file-space. The web server account requires full access to this directory, with the simplest method of ensuring this to give ownership to the directory. This global location is only suitable for ODS-5 volumes. Sixteen hexadecimal named subdirectories are used to partition metadata files with file names generated using data file full name escaped using extended parse syntax. Using this approach a sysadmin can easily locate specific metadata files if required.

   For example, with the global directive

   ```
   # WASD_CONFIG_GLOBAL
   [WebDAVmetaDir] DKA0:[WASDAVMETA]
   ```

   the data files

```
Directory WEB:[DAVweb]

01234^.56789.TXT;1    0.50KB   8-JUN-2009 23:07:19.26
0123456789.TXT;1      0.50KB   8-JUN-2009 23:06:59.16
```

would have the associated metadata files

```
Directory DKA0:[WASDAVMETA.06]

web^:^[davweb^]01234^.56789.txt__wasdav;1
                          1KB  19-JUN-2009 03:21:34.40
web^:^[davweb^]0123456789.txt__wasdav;1
                          1KB  19-JUN-2009 03:21:14.67
```

Directory Metadata

The metadata file associated with a directory is stored in the same metadata location as files contained by that directory (not in the metadata location associated with the parent directory that contains the directory file). This metadata file is named ".DIR_ _wasdav" (i.e. no name, just an extension), with the following example illustrating how this would appear in each of the three metadata locations, for a subdirectory named "New Folder".

```
WEB:[DAVweb.New^_Folder].DIR__wasdav;1
WEB:[DAVweb.New^_Folder.^.dav].DIR__wasdav;1
DKA0:[WASDAVMETA.06]web^:^[davweb^.new^_folder^].dir__wasdav;1
```

Metadata XML

All metadata is stored using XML. Multiple XML data can be contained in a single metadata file. Each can be individually manipulated by a WebDAV client. The property elements are stored as-supplied by the client. It is presumed that their XML well-formedness is guaranteed by the original request XML parsing. Metadata files have content similar to the following:

```
$ TYPE 0123456789.txt__wasdav;1
<?xml version="1.0" encoding="UTF-8"?>
<WASDAV:data xmlns:WASDAV="WASD.VMS.WebDAV"
updated="2009-06-18T17:49:14Z 19-JUN-2009 03:19:14">
<WASDAV:lock
token="opaquelocktoken:4D462D61B0E0427F19B425EBEEF2CFF6"
depth="0"
type="write"
scope="exclusive"
timeout="Second-86400"
expires="2009-06-20T22:49:14Z 21-JUN-2009 08:19:14">
<WASDAV:owner><NS:href xmlns:NS="DAV:">MGD</NS:href></WASDAV:owner>
</WASDAV:lock>
<WASDAV:prop>
<NS:one xmlns:NS="two">three</NS:one>
</WASDAV:prop>
<WASDAV:prop>
<NS:four xmlns:NS="five">six</NS:four>
</WASDAV:prop>
<WASDAV:prop>
<NS:seven xmlns:NS="eight">nine</NS:seven>
</WASDAV:prop>
</WASDAV:data>
```

This metadata example contains four properties; an exclusive write lock owned by "MGD" and three set by a client in three different (contrived) namespaces.

Metadata should not be edited manually . . .

. . . unless you really, really know what you're doing. WASD deletes meta-data files it does not understand or otherwise considers damaged (with some resultant loss of information). Of course you can, for example to remove a lock on a resource, but you run the (small) risk of a "lost-update" and other complications. And, again of course, full metadata can be deleted at the command-line.

Microsoft Metadata

An example of such property meta-data generated by a Microsoft Windows (not Internet) Explorer client (example wrapped for presentation):

```
<?xml version="1.0" encoding="UTF-8"?>
<WASDAV:data xmlns:WASDAV="WASD.VMS.WebDAV"
updated="2007-07-23T01:39:11Z">
<WASDAV:prop>
<NS:Win32CreationTime xmlns:NS="urn:schemas-microsoft-com:">
Tue, 26 Jun 2007 02:00:48 GMT</NS:Win32CreationTime>
</WASDAV:prop>
<WASDAV:prop>
<NS:Win32LastAccessTime xmlns:NS="urn:schemas-microsoft-com:">
Mon, 23 Jul 2007 01:52:32 GMT</NS:Win32LastAccessTime>
</WASDAV:prop>
<WASDAV:prop>
<NS:Win32LastModifiedTime xmlns:NS="urn:schemas-microsoft-com:">
Mon, 23 Jul 2007 01:52:32 GMT</NS:Win32LastModifiedTime>
</WASDAV:prop>
<WASDAV:prop>
<NS:Win32FileAttributes xmlns:NS="urn:schemas-microsoft-com:">
00000020</NS:Win32FileAttributes>
</WASDAV:prop>
</WASDAV:data>
```

Every file written or modified by *Windows Explorer* generates this sort of metadata which is then stored in an associated metadata file and read each time the data file is accessed. Some might consider this unnecessary clutter in most circumstances (I do). WASD allows this metadata to be suppressed and equivalent data generated (fudged) from file *live* properties when accessed - often sufficient for purpose. To suppress the actual processing of *Windows Explorer* metadata set a path using the WEBDAV=NOWINPROP in WASD_CONFIG_MAP.

```
set /webdav/* webdav=NOwinprop
```

6.4 WebDAV Locking

For efficiency and functionality considerations WebDAV locking may be enabled and disabled (default) as global functionality using the WASD_CONFIG_GLOBAL [WebDAVlocking] directive. Additionally the WEBVDAV=[NO]LOCKING path SETing can configure this on a per-path basis.

Write Access Only

In common with RFC 4918 WASD WebDAV locking controls only write access. Both exclusive and shared locks are provided. Locking applies to the DELETE, LOCK, MKCOL, MOVE, PROPPATCH, PUT, and UNLOCK methods.

Locking Depth

WASD WebDAV locking checks parent collections to a configurable depth. WASD_CONFIG_GLOBAL directive [WebDAVlockCollectionDepth] where the default (0 or 1) checks only WebDAV locking on files, 2 WebDAV locking on the parent directory, 3 on the grandparent, 4 the great-grandparent, etc. Of course each level can add significant latency (and expense) to some operations.

Lock Depth 0

> Real world experience has suggested locking depth should be maintained at the default 0 (or 1), allowing the client explicitly to manage and negotiate hierarchies of locking if required. WebDAV clients (probably correctly) assume a minimally compliant and relatively unsophisticated WebDAV server.

For more information on locking operation and implementation details see the DAVLOCK.C module and for meta-data in general the DAVMETA.C module.

Locking Timeout

When a client locks a resource it can specify the period for the lock. In the absence of such a specification WASD will apply the [WebDAVlockTimeoutDefault] value (by default 0-01:00:00 - one hour). WASD also applies the [WebDAVlockTimeoutMax] maximum lock period (by default 7-00:00:00 - one week). When the maximum period expires the lock is no longer valid.

VMS DLM Locking

WASD uses VMS locking to queue and arbitrate access to WebDAV resources and meta-files.

Two lock modes are employed; 'exclusive', when changes are to be made to the resource or its meta-data, and 'concurrent read', when resource and/or meta-data are only to be read. Concurrent read locks are compatible, but an exclusive queued against a resource currently being read waits, as does a read against a current exclusive.

WASD takes out its own VMS DLM locks on resources (files and directories) before beginning any WebDAV operation, and these prevent conflict with other WASD WebDAV operations on the same system or cluster, but RMS does not use these nor does WASD use RMS locks (except when actually acessing the file-system of course), and so there is potential for interactions between the two domains (in common with general file-system actvities). WASD WebDAV deliberately does not try to block file-system actions from other processing (except where RMS locks/blocks). Its own DLM locking is purely for internal purposes.

6.5 Some Wrinkles

Some application/environment-specific considerations when using WASD WebDAV. Please report any you encounter for future inclusion in this section. Also see Section 6.6 immediately below.

6.5.1 OS X Finder

OS X Finder requires [WebDAVlocking] enabled for read/write access, otherwise access will be read-only.

6.5.2 Gnome/gvfs/Nautilus

As at publication, *Gnome/gvfs/Nautilus* has quite a number of behavioural problems with associated Bugzilla items. Don't expect it to behave well! This has been my experience.

6.5.3 Dreamweaver

Dreamwever 8 (at least, the only version I have access to) insists on using a URI with a trailing "/./" occasionally (I'm guessing to specify the "current" directory - c.f. "/../", or "parent" syntax). Just absorb this internally using an appropriate mapping internal redirect.

```
redirect /webdav/**/./ /webdav/*/
```

6.6 Microsoft Miscellanea

A cornucopia of of minor and major considerations!

Microsoft approach WebDAV in their own inimitable fashion. Hence Microsoft agents, considering their ubiquity, including their mini-redirector are specifically looked for and functionality modified to accomodate them.

The following is a list topics/issues that were encountered/investigated during WASD WebDAV development. They may or may not be applicable to your site.

Some general references:

http://greenbytes.de/tech/webdav/webdav-redirector-list.html
http://greenbytes.de/tech/webdav/webfolder-client-list.html
http://www.zorched.net/2006/03/01/more-webdav-tips-tricks-and-bugs/
http://www.webdavsystem.com/server/documentation/troubleshooting
http://www.webdavsystem.com/documentation/troubleshooting
http://code.google.com/p/sabredav/wiki/Windows
http://ulihansen.kicks-ass.net/aero/webdav/
http://chapters.marssociety.org/webdav/

DOS/Windows command-line network configuration:

```
C:\> NET USE Z: http://the.host.name/folder/
C:\> NET USE Z: /DELETE
```

6.6.1 Mapping

Microsoft agents (at least) seem to request the server OPTIONS of the server root regardless of any path provided with the NET USE or other network drive mapping employed. To selectively map such a request into a path that has WebDAV enabled on it (and will therefore respond with the DAV-related options) use a conditional redirect rule. For example

```
if (webdav:)
   if (request-method:OPTIONS) redirect / /dav-path/
endif
```

or if only required for MS agents then something more specific

```
if (webdav:MSagent)
   if (request-method:OPTIONS) redirect / /dav-path/
endif
```

Subsequent rules will probably be required to map typeless directory requests to the actual directory required.

```
redirect /dav-path /dav-path/
pass /dav-path/* /dav_root/* webdav=read
```

6.6.2 FrontPage Extensions

Requests containing paths /_vti_inf.html and /_vti_bin/* are related to FrontPage protocol discovery probing. They can be adequately handled using a mapping rule lsuch as the following:

```
pass /_vti_* "404 Not an MS platform!"
```

6.6.3 Avoiding Microsoft Property Clutter

See Microsoft Metadata.

6.6.4 OPTIONS header "MS-Author-Via: DAV"

http://msdn2.microsoft.com/en-us/library/ms691698.aspx

If the server's response does not contain an MS-Author-Via header, the OLE DB Provider for Internet Publishing loads the WEC and WebDAV protocol drivers one at a time (WEC first, WebDAV second) and asks them, "Do you know how to handle this URL?", specifying the exact URL passed in by the client. The first protocol which responds "yes" is selected. If neither protocol driver responds "yes" then the method which triggered the automatic driver selection (usually IBindResource::Bind) fails with an OLE DB Provider for Internet Publishing specific error code IPP_E_SERVERTYPE_NOT_SUPPORTED.

6.6.5 Repairing broken XP Web Folders

http://chapters.marssociety.org/webdav/

Some Windows XP machines have a broken Web Folders installation. Microsoft includes a Web Folders repair utility built in to Windows to correct the problem. Use the following steps to fix the problem:

1. Click on the "Start" menu in the lower left corner, and select "Run..."
2. Type in "webfldrs.msi" and click the "OK" button.
3. Click on the "Select reinstall mode" button.
4. Select *ALL* of the checkboxes *except* for the second one ("Reinstall only if file is missing").
5. Click on the "OK" button.
6. Click on the "Reinstall" button.
7. After the reinstallation is complete, reboot the computer.

6.6.6 Adding a port number to the webfolder-address

Attach the port-number (80 by default) to the http-address you enter into the field of the "My Network Places"-assistant. As you can see in the following image and the linked screenshot, this will force Windows XP to use the "Microsoft Data Access Internet Publishing Provider DAV 1.1" mechanism instead of "Microsoft-WebDAV-MiniRedir/5.1.2600".

6.6.7 Adding a number-sign ("#") to the webfolder-address

It is also possible to add the number sign # to the http-address you enter into the field of the "My Network Places"-assistant. As you can see in the following image and the linked screenshot, this will also force Windows XP to use the "Microsoft Data Access Internet Publishing Provider DAV 1.1" mechanism instead of "Microsoft-WebDAV-MiniRedir/5.1.2600".

```
http://the.host.name/folder#
```

6.6.8 Force Windows XP to use Basic Authentication

There is a third way to get this working from the client-site. As described in the Microsoft Knowledge Base, Article ID: 841215, Windows XP disables "Basic Auth" in his "Microsoft-WebDAV-MiniRedir/5.1.2600"-mechanism by default for security reasons. See description below.

6.6.9 Microsoft XP Explorer BASIC Authentication

http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/sp2netwk.mspx

You can enable BasicAuth by adding the following registry key and setting it to a non-zero value:

```
HKEY_LOCAL_MACHINE\SYSTEM
\CurrentControlSet\Services\WebClient\Parameters\UseBasicAuth (DWORD)
```

If you delete the registry key or set it to 0, the behavior reverts to the default, or disabling the use of BasicAuth.

Disabling Basic Authentication over a clear channel:

Because the DAVRdr is part of the remote file-system stack, a computer is open to attack whenever an attempt is made to remotely access files. Although the threat to other applications that use the Internet APIs is less severe than it is for the DAVRdr, a similar attack is possible whenever an application (or the user) attempts to access a URL. For this reason, WinInet is exposing the mechanism by which the DAVRdr disables BasicAuth to other users of the Internet APIs.

With Windows XP Service Pack 2, there are two ways to block the use of Basic Authentication over clear (or unencrypted) channels:

Create the following registry key and set it to a non-zero value.

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion
\InternetSettings\DisableBasicOverClearChannel (DWORD)
```

This prevents WININET from attempting to use BasicAuth unless the channel is secured (HTTPS or SSL).

The application can disable the use of BasicAuth for its connections by setting the AUTH_FLAG_DISABLE_BASIC_CLEARCHANNEL flag (0x4) in the value supplied in the call to InternetSetOption using INTERNET_OPTION_AUTH_FLAGS.

*** **AND THEN RESTART WINDOWS** ***

6.6.10 Microsoft Windows 7 BASIC Authentication

You can enable BasicAuth by setting the following registry key to the value 3 and restarting the WebClient service:

```
HKEY_LOCAL_MACHINE\SYSTEM
\CurrentControlSet\Services\WebClient\Parameters\BasicAuthLevel (DWORD)
```

6.6.11 Error 0x800700DF: The file size exceeds the limit allowed and cannot be saved

"In my case I try to copy file over WEBDAV to WEB Client connection e.g. I have mapped drive to web site. file is about 70MB I can copy small files from the same WEBDav folder."

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WebClient\Parameters
```

1. Right click on the FileSizeLimitInBytes and click Modify
2. Click on Decimal
3. In the Value data box, type 4294967295, and then click OK. Note this sets the maximum you can download from the Webdav to 4 gig at one time, I havent figured out how to make it unlimited so if you want to download more you need to split it up.

   http://social.answers.microsoft.com/Forums/en/xphardware/thread/d208bba6-920c-4639-bd45-f345f462934f

Chapter 7

Proxy Services

A proxy server acts as an intermediary between Web clients and Web servers. It listens for requests from the clients and forwards these to remote servers. The proxy server then receives the responses from the servers and returns them to the clients. Why go to this trouble? There are several reasons, the most common being:

- To allow internal clients access to the Internet from behind a firewall. Browsers behind the firewall have full Web access via the proxy system.
- To provide controlled access to internal resources for external clients. The proxy server provides a managed gateway through a firewall into an organisation's Web resources.
- Many proxy servers provide caching, or local storage, of responses. For frequent or commonly accessed resources this can not only significantly reduce apparent network latency but also greatly reduce the total traffic downloaded by a site.
- For anonymity. Although often related directly to firewall security considerations, it can also sometimes be an advantage to just not reveal the exact source of Web transactions from within your local network.

Proxy Serving Quick-Start

No additional software needs to be installed to provide proxy serving.

Proxy servering is essentially configured using a combination of configuration directives in WASD_CONFIG_GLOBAL and WASD_CONFIG_SERVICE to enable proxy serving both globally and then for allow a specific service to make outgoing connections, along with mapping directives in WASD_CONFIG_MAP to control and direct those outgoing connections.

The following steps provide a brief outline of proxy configuration.

1. Enable proxy serving and specify which particular services are to be proxies (Section 7.1.1 and "WASD Web Services - Install and Config"),
2. If proxy caching is required (most probably, see Section 7.2)
    - Decide on a cache device, create the cache root directory, modify server startup procedures to include the WASD_CACHE_ROOT logical name (Section 7.2.1).

- Enable caching on required services (Section 7.2.2).
- Adjust relevant cache management configuration parameters if required (Section 7.2.3).
- If required adjust cache retention parameter (Section 7.2.5).

3. If providing SSL tunneling (proxy of Secure Sockets Layer transactions) add/modify a service for that (Section 7.3).
4. Add WASD_CONFIG_MAP mapping rules for controlling this/these services (Section 7.1.5, Section 7.3.2, and Section 7.4).
5. Restart server (HTTPD/DO=RESTART).

Error Messages

When proxy processing is enabled and WASD_CONFIG_GLOBAL directive [ReportBasicOnly] is disabled it is necessary to make adjustments to the contents of the WASD_CONFIG_MSG message configuration file [status] item beginning "Additional Information". Each of the "/httpd/-/status*nxx*.html" links

```
<A HREF="/httpd/-/status1xx.html">1<I>xx</I></A>
<A HREF="/httpd/-/status2xx.html">2<I>xx</I></A>
<A HREF="/httpd/-/status3xx.html">3<I>xx</I></A>
<A HREF="/httpd/-/status4xx.html">4<I>xx</I></A>
<A HREF="/httpd/-/status5xx.html">5<I>xx</I></A>
<A HREF="/httpd/-/statushelp.html">Help</A>
```

should be changed to include a local host component

```
<A HREF="http://local.host.name/httpd/-/status1xx.html">1<I>xx</I></A>
<A HREF="http://local.host.name/httpd/-/status2xx.html">2<I>xx</I></A>
<A HREF="http://local.host.name/httpd/-/status3xx.html">3<I>xx</I></A>
<A HREF="http://local.host.name/httpd/-/status4xx.html">4<I>xx</I></A>
<A HREF="http://local.host.name/httpd/-/status5xx.html">5<I>xx</I></A>
<A HREF="http://local.host.name/httpd/-/statushelp.html">Help</A>
```

If this is not provided the links and any error report will be interpreted by the browser as relative to the server the proxy was attempting to request from and the error explanation will not be accessible.

7.1 HTTP Proxy Serving

WASD provides a proxy service for the HTTP scheme (prototcol).

Proxy serving generally relies on DNS resolution of the requested host name. DNS lookup can introduce significant latency to transactions. To help ameliorate this WASD incorporates a host name cache. To ensure cache consistency the contents are regularly flushed, after which host names must use DNS lookup again, refreshing the information in the cache. The period of this cache purge is contolled with the [ProxyHostCachePurgeHours] configuration parameter.

When a request is made by a proxy server is is common for it to add a line to the request header stating that it is a forwarded request and the agent doing the forwarding. With WASD proxying this line would look something like this:

```
Forwarded: by http://host.name.domain (HTTPd-WASD/8.4.0 OpenVMS/IA64 SSL)
```

It is enabled using the [ProxyForwarded] configuration parameter.

An additional, and perhaps more widely used facility, is the Squid extension field to the proxied request header supplying the originating client host name or IP address.

```
X-Forwarded-For: client.host.name
```

It is enabled using the [ProxyXForwardedFor] configuration parameter.

7.1.1 Enabling A Proxy Service

Proxy serving is enabled on a global basis using the WASD_CONFIG_GLOBAL file [Proxy-Serving] configuration parameter. After that each virtual service must have proxy functionality enabled as a per-service configuration.

WASD can configure services using the WASD_CONFIG_GLOBAL [service] directive, the WASD_CONFIG_SERVICE configuration file, or even the /SERVICE= qualifier.

WASD_CONFIG_SERVICE

Using directives listed in "WASD Web Services - Install and Config" this example illustrates configuring a non-proxy server (the *disabled* is the default and essentially redudant) and a proxy service.

```
[[http://alpha.example.com:80]]
[ServiceProxy]  disabled

[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
```

7.1.2 Proxy Affinity

High performance/highly available proxy server configurations require more than one instance configured and running. Whether this is done by running multiple instances on the same host or one instance on multiple hosts, it leads to situations where successive requests will be processed by different instances. As those instances don't share a common name to IP address cache, they will eventually use different IP addresses when trying to connect to an origin server running on multiple hosts.

This may result in the following, user visible, issues:

- multiple requests for authentication (one from each origin host)
- loss of icons, images, javascripts, CSS because requests for these files, although they return a 401 status, will not trigger a browser authentication dialog
- loss of context and performance issues where scripts/environments need to be started on a new host (php, python, webware,...)

For these reasons, the proxy server will make every effort to relay successive requests from a given client to the same origin host as long as this one is available (built-in failover capability will ultimately trigger the choice of a new host). This is known as client to origin affinity or proxy affinity capability.

Proxy to origin server affinity is enabled using the following service configuration directive.

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyAffinity]  enabled
```

Uses HTTP Cookies

Obviously the use of cookies must be enabled in the browser or this facility will not operate for that client. After the first successful connection to an origin host, the proxy server will send a cookie indicating the IP address used to the client browser. Upon subsequent requests, this cookie will be used to select the same host. The cookie is named *WasdProxyAffinity_origin.host.name* and the value simply the IP address in dotted decimal. This cookie is not propagated beyond the proxy service but may be WATCHed by checking the *Proxy Processing* item.

7.1.3 Proxy Bind

It is possible to make the outgoing request appear to originate from a particular source address. The Network Interface must be able to bind to the specified IP address (i.e. it cannot be an arbitrary address).

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyBind]  131.185.250.1
```

The same behaviour may be accomplished with an WASD_CONFIG_MAP mapping rule.

```
SET http://*.example.com proxy=bind=131.185.250.1
```

7.1.4 Proxy Chaining

Some sites may already be firewalled and have corporate proxy servers providing Internet access. It is quite possible to use WASD proxying in this environment, where the WASD server makes the proxied requests via the next proxy server in the hierarchy. This is known as *proxy chaining*.

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyChain]  next.proxy.host
```

Chaining may also be controlled on a virtual service or path basis using an WASD_CONFIG_MAP mapping rule.

```
SET http://*.com proxy=chain=next.proxy.host:8080
```

Chain Authorization

If the upstream proxy server requires authorization this may be supplied using a per-service directive

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyChain]  next.proxy.host
[ServiceProxyChainCred]  <username>:<password>
```

or via mapping rule

```
SET http://*.com proxy=chain=next.proxy.host:8080 \
                 proxy=chain=cred=<username>:<password>
```

7.1.5 Controlling Proxy Serving

Requests at a service enabled for proxy processing are directed to proxy processing using a fundamental rule which terminates rule processing and initiates the outgoing connection.

```
pass * http://
```

This rule and variant equivalents for FTP and CONNECT processing, and in combination with other rules to purpose, are seen in the examples in this section on proxy.

Controlling both access-to and access-via proxy serving is possible.

Proxy Password

Access to the proxy service can be directly controlled through the use of WASD authorization. Proxy authorization is distinct from general access authorization. It uses specific *proxy authorization* fields provided by HTTP, and by this allows a proxied transaction to also supply transaction authorization for the remote server. In the WASD_CONFIG_SERVICE configuration file.

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyAuth]  proxy
```

In addition to the service being specified as requiring authorization it is also necessary to configure the source of the authentication. This is done using the WASD_CONFIG_AUTH configuration file. The following example shows all requests for the proxy virtual service must be authorized (GET and well as POST, etc.), although it is possible to restrict access to only read (GET), preventing data being sent out via the server.

```
[[alpha.example.com:8080]]
["Proxy Access"=PROXY_ACCESS=id]
http://* read+write
```

Chain Password

An up-stream, chained proxy server (Section 7.1.4) may be permitted to receive proxy authentication from the client via a WASD proxy server using the CHAIN keyword. Unconfigured, WASD does not propagate HTTP *proxy authorization* fields. Only one proxy server in a chain can be authenticated against.

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyAuth]  chain
```

Local Password

It is also possible to control proxy access via local authorization, although this is less flexible by removing the ability to then pass authorization information to the remote service. In other repects it is set up in the same way as proxy authorization, but enabled using the LOCAL keyword.

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyAuth]  local
```

Access Filtering

Extensive control of how, by whom and what a proxy service is used for may be exercised using WASD general and conditional mapping “WASD Web Services - Install and Config” and “WASD Web Services - Install and Config” possibly in the context of a virtual service specification for the particular connect service host and port (see “WASD Web Services - Install and Config”), The following examples provide a small indication of how mapping could be used in a proxy service context.

1. It is possible, though more often not practical, to regulate which hosts are connected to via the proxy service. For example, the following rule forbids accessing any site with the string “hacker” in it (for the proxy service “alpha . . . :8080”.

   ```
   [[alpha.example.com:8080]]
   pass http://*hacker*/* "403 Proxy access to this host is forbidden."
   pass http://*
   ```

2. Or as in the following example, only allow access to specific sites.

   ```
   [[alpha.example.com:8080]]
   pass http://*.org/*
   pass http://*.digital.com/*
   pass http://* "403 Proxy access to this host is forbidden."
   ```

3. It is also possible to restrict access via the proxy service to selected hosts on the internal subnet. Here only a range of literal addresses plus a single host in another subnet are allowed access to the service.

   ```
   [[alpha.example.com:8080]]
   pass http://* "403 Restricted access." ![ho:131.185.250.* ho:131.185.200.10]
   pass http://*
   ```

4. In the following example POSTing to a particular proxied servers is not allowed (why I can't imagine, but hey, this is an example!)

```
[[alpha.example.com:8080]]
pass http://subscribe.sexy.com/* "403 POSTing not allowed." [me:POST]
pass http://*
```

5. It is possible to redirect proxied requests to other sites.

```
[[alpha.example.com:8080]]
redirect http://www.sexy.com/* http://www.disney.com/
pass http://*
```

6. A proxy service is just a specialized capability of a general HTTP service. Therefore it is quite in order for the one service to respond to standard HTTP requests as well as proxy-format HTTP requests. To enforce the use of a particular service as proxy-only, add a final rule to a virtual service's mapping restricting non-proxy requests.

```
[[alpha.example.com:8080]]
pass http://*
pass /* "403 This is a proxy-only service."
```

7. This example provides the essentials when supporting *reverse proxying*. Note that mappings may become quite complex when supporting access to resources across multiple internal systems (e.g. access to directory icons).

```
[[main.corporate.server.com:80]]
pass /sales/* http://sales.corporate.server.com/*
pass /shipping/* http://shipping.corporate.server.com/*
pass /support/* http://support.corporate.server.com/*
pass * "403 Nothing to access here!"
```

Note

To expedite proxy mapping is it recommended to have a final rule for the proxy virtual service that explicitly *pass*es the request. This would most commonly be a permissive pass as in example 1, could quite easily be an restrictive pass as in example 2, or a combination as in example 6.

7.2 Caching

Caching involves using the local file-system for storage of responses that can be reused when a request for the same URL is made. The WASD server does not have to be configured for caching, it will provide proxied access without any caching taking place.

When a proxied request is processed, and the characteristics would allow the response to be cached, a unique identifier generated from the URL is used to create a corresponding file name. The response header and any body are stored in this file. This may be the data of an HTML page, a graphic, etc.

When a proxied request is being processed, and the characteristics would allow the request to be cached, the unique identifier generated allows for a previously created cache file to be checked for. If it exists, and is current enough, the response is returned from it, instead of from the remote server. If it exists and is no longer current the request is re-made to the remote server, and the response if still cacheable is re-cached, keeping the contents current. If it does not exist the response is delivered from the remote server.

Not all responses can be cached!

The main critera are for the response to be successful (200 status), general (i.e. one not in response to a specialized query or action), and not too volatile (i.e. the same page may be expected to be returned more than once, preferably over an extended period).

- Proxied *requests* can only be cached if . . .
  - uses the GET method
  - does not contain a query string
  - is HTTP/1.n compliant (i.e. not HTTP/0.9)
  - does not contain an "Authorization:" header field
- Proxied *success responses* will only be cached if . . .
  - is HTTP/1.n compliant (i.e. not HTTP/0.9)
  - HTTP status code 200 (success), 203 (non-authoritative), 300 (multiple choice), 301 (moved permanently), 410 (gone)
  - contains a *Last-Modified:* header field
  - one or more hours since the last modification
  - any *Expires:* date/time is still in the future
  - does not contain restrictive cache control
    “Pragma: no-cache” field (HTTP/1.0)
    “Cache-Control: no-cache, no-store, private” (/1.1)
  - any “Vary:” header field does not contain a “*” or “accept[-...]"”
  - does not exceed a configuration parameter in size
- Proxied *negative responses* will be cached if . . .
  - [ProxyCacheNegativeSeconds] is non-zero
  - status code 204 (no content), 305 (use proxy), 400 (bad request), 403 (forbidden), 404 (not found), 405 (method not allowed), 414 (request URI too large), 500 (internal server error), 501 (not implemented), 502 (bad gateway), 503 (service unavailable), 504 (gateway timeout),
  - does not contain restrictive cache control
    “Pragma: no-cache” field (HTTP/1.0)
    “Cache-Control: no-cache, no-store, private” (/1.1)

The [ProxyCacheFileKbytesMax] configuration parameter controls the maximum size of a response before it will not be cached. This can be determined from any “Content-Length:” response header field, in which case it will proactively not be cached, or if during cache load the maximum size of the file increases beyond the specified limit the load is aborted.

Not all sites may benefit from cache!

As many transactions on today's Web contain query strings, etc., and therefore cannot be meaningfully cached, it should not be assumed the cost/benefit of having a proxy cache enabled is a forgone conclusion. Each site should monitor the proxy traffic reports and decide on a local policy.

The facilities described in Section 7.2.6 allow a reasonably informed decision to be made. Items to be considered.

- The ratio of cache reads to network accesses.
- The number of non-cacheable requests and responses, particularly as a percentage of total proxy traffic.
- The ratio of network to cache traffic, although this may be skewed by having a high ratio of 304 (not-modified) responses from cache (which contain few bytes). Check the cache 304 reporting item.

Last, but by no means least, understanding the characteristics of local usage. For example, are there a small number of requests generating lots of non-cacheable traffic? For instance, a few users accessing streaming content.

7.2.1 Cache Device

Selection of a disk device for supporting the proxy cache should not be made without careful consideration, doubly so if significant traffic is experienced. Here are some common-sense suggestions.

- **avoid** locating it as a subdirectory of WASD_ROOT:[000000]
- use a disk with as little other activity as possible (both I/O and space usage)
- use a disk with as much free space as possible
- use the fastest disk available

Initially the directory will need to be created. This can be done manually as described below, or if using the supplied server startup procedures (see "WASD Web Services - Install and Config") it is checked for and if it does not exist is automatically created during startup. The directory must be owned by the HTTP$SERVER account and have full read+write+execute+delete access. It is suggested to name it [WASD_CACHE] and may be created manually using the following command.

```
$ CREATE /DIR /OWN=HTTP$SERVER /PROT=(O:RWED,G,W) device:[WASD_CACHE]
```

It is a relatively simple matter to relocate the cache at any stage. Simply create the required directory in the new location, modify the startup procedures to reflect this, shut the server down completely then restart it using the procedures (**not** a /DO=RESTART!). The contents of the previous location could be transfered to the new using the BACKUP utility if desired.

WASD_CACHE_ROOT Logical

It is required to define the logical name WASD_CACHE_ROOT if any proxy services are specified as using cache in the server configuration. The server will not start unless it is correctly defined. The logical should be a *concealed device* logical specifying the top level directory of the cache tree. The following example shows how to define such a logical name.

```
$ DEFINE /SYSTEM /EXEC /TRANSLATION=CONCEALED WASD_CACHE_ROOT device:[WASD_CACHE.]
```

If example startup procedure is in use then it is quite straight-forward to have the logical created during server startup (see "WASD Web Services - Install and Config").

7.2.2 Enabling Caching

Caching may enabled on a per-service basis. This means it is possible to have a caching proxy service and a non-caching service active on the one server. Caching is enabled by appending the *cache* keyword to the particular service specification. The following example shows a non-proxy and a caching proxy service.

```
[[http://alpha.example.com:80]]
[ServiceProxy]  disabled

[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyCache]  enabled
```

Proxy caching may be selectively disabled for a particular site, sites or paths within sites using the *SET nocache* mapping rule. This rule, used to disable caching for local requests, also disables proxy file caching for that subset of requests. This example shows a couple of variations.

```
[[alpha.example.com:8080]]
# disable caching for local site's servers that respond fairly quickly
set http://*.local.domain/* nocache
# disable caching of log files
set http://*.log nocache
pass http://*
```

Note

It is also recommended to place the cache directory under some authorization control to prevent casual browsing and access of the cache contents. Something local, similar in intention to

```
[[alpha.example.com:8080]]
["WASD Admin"=WASD_ADMIN=id]
/wasd_cache_root/* ~webadmin,131.185.250.*,r+w ;
```

7.2.3 Cache Management

As the proxy cache is implemented using the local file system, management of the cache implies controlling the number of, and exactly which files remain in cache. Essentially then, management means when and which to delete. The [ProxyReportLog] configuration parameter enables the server process log reporting of cache management activities.

Cache file deletion has three variants.

1. **ROUTINE**

This ensures files that have not been accessed within specified limits are periodically and regularly deleted. The [ProxyCacheRoutineHourOfDay] configuration parameter controls this activity.

The ROUTINE form occurs once per day at the specified hour. The cache files are scanned looking for those that exceed the configuration parameter for maximum period since last access, which are then deleted (the largest number of [ProxyCachePurgeList], as described below).

2. **BACKGROUND**

   Setting the [ProxyCacheRoutineHourOfDay] configuration parameter to 24 enables background purging.

   In this mode the server continuously scans through the cache files in the same manner as for ROUTINE purging. The difference is it is not all done a single burst once a day, pushing disk activity to the maximum. The background purge regulates the period between each file access, pacing the scan so that the entire cache is passed through once a day. It adjusts this pace according the the size of the cache.

3. **REACTIVE**

   This is a remedial action, when cache device usage is reaching its configuration limit and files need to be deleted to free up space. The following parameters control this behaviour.

   [ProxyCacheDeviceCheckMinutes]
   [ProxyCacheDeviceMaxPercent]
   [ProxyCacheDevicePurgePercent]
   [ProxyCachePurgeList]

   The cache device space usage is checked at the specified interval.

   If the device reaches the specified percentage used a cache purge is initiated and by deleting files until the specified reduction is attained, the total space in use on the disk is reduced.

   The cache files are scanned using the [ProxyCachePurgeList] parameter described below, working from the greatest to least number of hours in the steps provided. At each scan files not accessed within that period are deleted. At each few files deleted the device free space is checked as having reached the lower purge percentage limit, at which point the scan terminates.

   This parameter has as its input a series of comma-separated integers representing a series of hours since files were last accessed. In this way the cache can be progressively reduced until percentage usage targets are realized. Such a parameter would be specified as follows,

```
[ProxyCachePurgeList] 168,48,24,8,0
```

   meaning the purge would first delete files not accessed in the last week, then not for the last two days, then the last twenty-four hours, then eight, then finally all files. The largest of the specified periods (in this case 168) is also used as the limit for the ROUTINE scan and file delete.

Once the target reduction percentage is reached the purge stops. During the purge operation further cache files are not created. Even when cache files cannot be created for any reason proxy serving still continues transparently to the clients.

Note

Cache files can be manually deleted at any time (from the command line) without disturbing the proxy-caching server and without rebuilding any databases. When deleting, the /BEFORE=*date*/*time* qualifier can be used, with /CREATED being the document's last-modified date, /REVISED being the last time it was loaded, and /EXPIRED the last time the file was accessed (used to supply a request). Be aware that on an active server it is quite possible some files may be locked at time of attempted deletion.

From The Command-Line

If [ProxyCacheRoutineHourOfDay] is empty or non-numeric the automatic, once-a-day routine purge of the cache by the server is disabled and it is expected to be performed via some other mechanism, such as a periodic batch job. This allows routine purging more or less frequently than is provided-for by server configuration, and/or the purge activity being performed by a process or cluster node other than that of the HTTPd server (reducing server and/or node impact of this highly I/O intensive activity). Progress and other messages are provided via SYS$OUTPUT, and if configured in the [Opcom . . .] directives to the operator log and designated operator terminal as well. If a process already has the cache locked the initiated activity aborts.

The following example shows a routine purge being performed from the command-line. This form uses the hours from [ProxyCachePurgeList].

```
$ HTTPD /PROXY=PURGE=ROUTINE
```

A variant on this allows the maximum age to be explicitly specified.

```
$ HTTPD /PROXY=PURGE=ROUTINE=168
```

Reactive purging and statistic scans may also be initiated from the command line. For a reactive purge the first number can be the device usage percentage (indicated by the trailing "*%*"), if not the configuration limit is used.

```
$ HTTPD /PROXY=PURGE=REACTIVE=80%,168,48,24,8,0
$ HTTPD /PROXY=CACHE=STATISTICS
```

Any in-progress scan of the cache (i.e. reactive or routine purges, or a statistics scan) can be halted from the command line (and online Server Admininistration facility).

```
$ HTTPD /PROXY=STOP=SCAN
```

7.2.4 Cache Invalidation

For the purposes of this document, cache invalidation is defined as the determination when a cache file's data is no longer valid and needs to be reloaded.

The method used for cache validation is deliberately quite simple in algorithm and implementation. In this first attempt at a proxy server the overriding criteria have been efficiency, simplicity of implementation, and reliability. Wishing to avoid complicated revalidation using behind-the-scenes HEAD requests the basic approach has been to just invalidate the cache item upon exiry of a period related to the “Last-Modified:” age or upon a *no-cache* request, both described further below.

- If a “Pragma: no-cache” request header field is present (as is generated by Netscape Navigator when using the *reload* function) then the server should completely reload the response from the remote server. (Too often the author seems to have received incomplete responses where the proxy server caches only part of a response and has seemed to refuse to explicitly re-request.) OK, it’s a bit more expensive but who’s to say the proxy server is right all the time! The response is still cached ... the next request may not have the *no-cache* parameter.
- When a response is cached the file *creation* date/time is set to the local equivalent of the “Last-Modified:” GMT date and time supplied with the response. In this manner the file’s absolute age can be determined quickly and easily from the file header. This is used as described in Section 7.2.5.
- When a file is cached, the *revision* and *expires* date/times are set to current. The revision date/time is used when assessing when the file was last loaded/validated/reloaded. Once a file is cached the RMS *expires* date/time is updated every time it is subsequently accessed. In this way recency of usage of the item can be easily tracked, allowing the routine and reactive purges to operate by merely checking the file header.

The *revision count* (automatically updated by VMS) tracks the absolute number of accesses since the file was created (actually a maximum of 65535, or an unsigned short, but that should be enough for informational purposes).

7.2.5 Cache Retention

The [ProxyCaheReloadList] configuration parameter is used to control when a file being accessed is reloaded from source.

This parameter supplies a series of integers representing the hours after which an access to a cache file causes the file to be invalidated and reloaded from the source during the proxied request. Each number in the series represents the lower boundary of the range between it and the next number of hours. A file with a last-loaded age falling within a range is reloaded at the lower boundary of that particular range. The following example

```
[ProxyCacheReloadList] 1,2,4,8,12,24,48,96,168
```

would result in a file 1.5 hours old being reloaded every hour, 3.25 hours old every 2 hours, 7 hours old every 4 hours, etc. Here "old" means since last (or of course first) loaded. Files not reloaded since the final integer, in this example 168 (one week), are always reloaded.

7.2.6 Reporting and Maintenance

The HTTPDMON utility allows real-time monitoring of proxy serving activity (Section 13.9).

Proxy reports and some administrative control may be exercised from the online Server Administration facility (Chapter 9). The information reported includes:

- some proxy serving statistics
- current cache device status
- whether cache space is available
- if a purge is in progress
- the results from the last routine and reactive purges
- the results from the last scan of the cache
- contents of the host name/address cache

The following actions can be initiated from this menu. Note that three of these relate to proxy file cache and so may take varying periods to complete, depending on the number of files. If the cache is particularly large the scan/purge **may take some considerable time**.

- generate proxy cache statistics by scanning the entire cache
- perform a routine purge
- perform a reactive purge
- purge the proxy host name/address cache

Also available from the Server Administration facility is a dialog allowing the proxy characteristics of the *running* server to be adjusted on an ad hoc basis. This only affects the executing server, to make changes to permanent configuration the WASD_CONFIG_GLOBAL configuration file must be changed.

This dialog can be used to modify the device free space percentages according to recent changes in device usage, alter the reload or purge hour list characteristics, etc. After making these changes a routine or reactive purge will automatically be initiated to reduce the space in use by the proxy cache if implied by the new settings.

7.2.7 PCACHE Utility

It is often useful to be able to list the contents of the proxy cache directory or the characteristics or contents of a particular cache file. Cache files have a specific internal format and so require a tool capable of dealing with this. The WASD_ROOT:[SRC.UTILS]PCACHE.C program provides a versatile command-line utility as well as CGI(plus) script, making cache file information accessible from a browser. It also allows cache files to be selected by wildcard filtering on the basis of the contents of the associated URL or response header. For detailed information on the various command-line options and CGI query-string options see the description at the start of the source code file.

Command-Line Use

Make the WASD_EXE:PCACHE.EXE executable a foreign verb. It is then possible to

- list the basic characteristics of all/selected files in the cache directory tree
- list the characteristics plus the HTTP response header of a single file
- extract the response header
- extract the response body (text, graphic, file, etc.)
- do all of the above while filtering on URL or response header contents, number of hits, when last accessed, last loaded, and last modified (in hours)

Script Use

To make the PCACHE script available to the server ensure the following line exists in the HTTP$CONFIG configuration file in the [AddType] section.

```
.HTC  application/x-script  /cgiplus-bin/pcache  WASD proxy cache file
```

The following rule needs to be in the WASD_CONFIG_MAP configuration file.

```
pass /wasd_cache_root/*
```

Note

It is also recommended to place the utility and the cache directory under some authorization control to prevent casual browsing and access of the cache contents. Something local, similar in intention to

```
[[alpha.example.com:8080]]
["WASD Admin"=WASD_ADMIN=id]
/pcache/* ~webadmin,131.185.250.*,r+w ;
/wasd_cache_root/* ~webadmin,131.185.250.*,r+w ;
```

Once available the following is then possible.

- From a directory listing (“Index Of”) access a cache file and be presented with the following information:
  - blocks used/allocated
  - last modification date/time of the response
  - date/time the response was (re)loaded into cache
  - date/time the cache file was last accessed
  - number of time since first created the cache file has been accessed
  - the URL the cache file represents (as a link)
  - the full response header (as received from the proxied server)
  - a series of “buttons” allowing
    - the cache content (response body) to be viewed (note that self-relative embedded graphics, etc., probably will not be displayed in such documents)

- the cache file to be VMS DUMPed
- the cache file to be VMS ANALYZE/RMSed
- the cache file to be VMS DELETEd

If the configuration changes described above have been made the following link will return such an index.

online hypertext link

- Have the utility generate a form providing a convenient interface to the various capabilities and filters available. If the configuration changes described above have been made the following link will return this form.

  online hypertext link

- The utility's form does not have to be used. By supplying the appropriate query string components, either from a custom form or forms, or directly embedded into links, profiles, listings, deletion may be generated.

Note

Cache directory trees have the potential to become heavily populated, so the use of the script to generate listings of the cache contents could return extremely large listing documents.

7.3 CONNECT Serving

The *connect* service provides firewall proxying for any connection-oriented TCP/IP access. Essentially it provides the ability to tunnel any other protocol via a Web proxy server. In the context of Web services it is most commonly used to provide firewall-transparent access for Secure Sockets Layer (SSL) transactions. It is a special case of the more general tunneling provided by WASD, see Section 7.6.

7.3.1 Enabling CONNECT Serving

As with proxy serving in general, CONNECT serving may enabled on a per-service basis using the WASD_CONFIG_GLOBAL [service] directive, the WASD_CONFIG_SERVICE configuration file, or even the /SERVICE= qualifier.

The actual services providing the CONNECT access (i.e. the host and port) are specified on a per-service basis. This means it is possible to have CONNECT and non-CONNECT services deployed on the one server, as part of a general proxy service or standalone. CONNECT proxying is enabled by appending the *connect* keyword to the particular service specification. The following example shows a non-proxy and proxy services, with and without additional connect processing enabled.

```
[[http://alpha.example.com:80]]

[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled

[[http://alpha.example.com:8081]]
[ServiceProxyTunnel]  connect
```

```
[[http://alpha.example.com:8082]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  connect
```

7.3.2 Controlling CONNECT Serving

The connect service poses a significant security dilemma when in use in a firewalled environment. Once a CONNECT service connection has been accepted and established it essentially acts as a relay to whatever data is passed through it. Therefore **any transaction whatsoever** can occur via the connect service, which in many environments may be considered undesirable.

In the context of the Web and the use of the connect service for proxying SSL transactions it may be well considered to restrict possible connections to the well-known SSL port, 443. This may be done using conditional directives, as in the following example:

```
[[alpha.example.com:8080]]
if (request-method:CONNECT)
   pass *:443
   pass * "403 CONNECT only allowed to port 443."
endif
```

All of the comments on the use of general and conditional mapping made in Section 7.1.5 can also be applied to the connect service.

7.4 FTP Proxy Serving

WASD provides a proxy service for the FTP scheme (prototcol). This provides the facility to list directories on the remote FTP server, download and upload files.

The (probable) file system of the FTP server host is determined by examining the results of an FTP PWD command. If it returns a current working directory specification containing a "/" then it assumes it to be Unix(-like), if ":[" then VMS, if a "\" then DOS. (Some DOS-based FTP servers respond with a Unix-like "/" so a second level of file-system determination is undertaken with the first entry of the actual listing.) Anything else is unknown and reported as such. WASD (for the obvious reason) is particularly careful to perform well with FTP servers responding with VMS file specifications.

Note that the content-type of the transfer is determined by the way the proxy server interprets the FTP request path's "file" extension. This may or may not correspond with what the remote system might consider the file type to be. The default content-type for unknown file types is "application/octet-stream" (binary). When using the *alt* query string parameters then for any file in a listing the icon provides an alternate content-type. If the file link provides a text document then the icon will provide a binary file. If the link returns a binary file then the icon will return a file with a plain-text content-type.

In addition to content-type the FTP mode in which the file transfer occurs can be determined by either of two conditions. It the content-type is "text/.." then the transfer mode will be ASCII (i.e. record carriage-control adjusted between systems). If not text then the file is transfered in Image mode (i.e. a binary, opaque octet-stream). For any given content-type this default behaviour may be adjusted using the [AddType] directive (see "WASD Web Services - Install and Config") or the "#!+" MIME.TYPES directive (see "WASD Web Services - Install and Config").

Rules required in WASD_CONFIG_MAP for mapping FTP proxy. This is preferably made against the virtual service providing the FTP proxy. The service explicitly must make the icon path used available or it must be available to the proxy service in some other part of the mappings. Also the general requirement for error message URLs applies to FTP proxying (Error Messages).

```
[[proxy.host.name:8080]
pass http://* http://*
pass ftp://* ftp://*
pass /*/-/* /wasd_root/runtime/*/*
```

7.4.1 FTP Query String Keywords

Keywords added to an FTP request query string allow the basic FTP action to be somewhat tailored. These case-insensitive keywords can be in the form of a query keys or query form fields and values. This allows considerable flexibility in how they are supplied, allowing easy use from a browser URL field or for inclusion as form fields.

FTP Query String Keywords

Keyword	Description
alt	Adds alternate access (complementary content-type at the icon) for directory listings.
ascii	Force the file transfer type to be done as ASCII (i.e. with carriage-control conversion between systems with different representations).
content	Explicitly specify the content type for the returned file (e.g. “content:text/plain”, or “content=image/gif”).
dos	When generating a directory listing force the interpretation to be DOS.
email	Explicitly specify the *anonymous* access email address (e.g. “email:daniel@wasd.vsm.com.au” or “email=daniel@wasd.vsm.com.au”).
image	Force the file transfer type to be done as an opaque binary stream of octets.
list	Displays the actual directory plain-text listing returned by the remote FTP server. Can be used for problem analysis.
login	Results in the server prompting for a username and password pair that are then used as the login credentials on the remote FTP server.
octet	Force the content-type of the file returned to be specified as "application/octet-stream".
text	Force the content-type of the file returned to be specified as "text/plain".
unix	When generating a directory listing force the interpretation to be Unix.
upload	Causes the server to return a simple file transfer form allowing the upload of a file from the local system to the remote FTP server.
vms	When generating a directory listing force the interpretation to be VMS.

7.4.2 "login" Keyword

The usual mechanism for supplying the username and password for access to a non-anonymous proxied FTP server area is to place it as part of the request line (i.e. "ftp://username:password@the.host.name/path/"). This has the obvious disadvantage that it's there for all and sundry to see.

The "login" query string is provided to work around the more obvious of these issues, having the authentication credentials as part of the request URL. When this string is placed in the request query string the FTP proxy requests the browser to prompt for authentication (i.e. returns a 401 status). When request header authentication data is present it uses this as the remote FTP server username and password. Hence the remote username and password never need to appear in plain-text on screen or in server logs.

7.5 Gatewaying Using Proxy

WASD is fully capable of mapping non-proxy into proxy requests, with various limitations on effectiveness considering the nature of what is being performed.

Gatewaying between request schemes (protocols)

HTTP to HTTP (a gateway *of sorts* - standard proxy)
HTTP TO HTTP-over-SSL (non-secure to secure)
HTTP to FTP
HTTP-over-SSL to HTTP (secure to non-secure)
HTTP-over-SSL to HTTP-over-SSL (secure to secure)
HTTP-over-SSL to FTP

and also gatewaying between IP versions

IPv4 to IPv6
IPv6 to IPv4

All can be useful for various reasons. One example might be where a script is required to obtain a resource from a secure server via SSL. The script can either be made SSL-aware, sometimes a not insignificant undertaking, or it can use standard HTTP to the proxy and have that access the required server via SSL. Another example might be accessing an internal HTTP resource from an external browser securely, with SSL being used from the browser to the proxy server, which the accesses the internal HTTP resource on its behalf.

Request Redirect

The basic mechanism allowing this gatewaying is "internal" redirection. The *redirect* mapping rule (see "WASD Web Services - Install and Config") either returns the new URL to the originating client (requiring it to reinitiate the request) or begins reprocessing the request internally (transparently to the client). It is this latter function that is obviously used for gatewaying.

7.5.1 Reverse Proxy

The use of WASD proxy serving as a firewall component assumes two configured network interfaces on the system, one of which is connected to the internal network, the other to the external network. (Firewalling could also be accomplished using a single network interface with router blocking external access to all but the server system.) Outgoing (internal to external) proxying is the most common configuration, however a proxy server can also be used to provide controlled external access to selected internal resources. This is sometimes known as *reverse proxy* and is a specific example of WASD's general *non-proxy to proxy* request redirection capability (Section 7.5).

In this configuration the proxy server is contacted by an external browser with a standard HTTP request. Proxy server rules map this request onto a proxy-request format result. For example:

```
redirect /sales/* /http://sales.server.com/*?
```

Note that the trailing question-mark is required to propagate any query string (see "WASD Web Services - Install and Config").

The server recognises the result format and performs a proxy request to a system on the internal network. Note that the mappings required could become quite complex, but it is possible. See example 7 in Section 7.1.5.

Redirection Location Field

If a reverse proxied server returns a redirection response (302) containing a "Location: *url*" field with the host component the same reverse-proxied-to server it can be rewritten to instead contain the proxy server host. If these do not match the rewrite does not occur. Using the redirection example above, the SET mapping rule *proxy=reverse=location* specifies the path that will be prefixed to the path component in the location field URL. Usually this would be the same path used to map the reverse proxy redirect (in this example "/sales/"), though could be any string (presumably detected and processed by some other part of the mapping).

```
set /sales/* proxy=reverse=location=/sales/
redirect /sales/* /http://sales.server.com/*?
```

This could be simplified a little by using a postfix SET rule along with the original redirect.

```
redirect /sales/* /http://sales.server.com/*? proxy=reverse=location=/sales/
```

If the *proxy=reverse=location=<string>* ends in an asterisk the entire 302 location field URL is appended (rather than just the path) resulting in something along the lines of

```
Location: http://proxy.server.com/sales/http://sales.server.com/path/
```

which once redirected by the client can be subsequently tested for and some action made by the proxy server according to the content (just a bell or whistle ;-).

Authorization Verification

WASD can authorize reverse proxy requests locally (perhaps from the SYSUAF) and rewrite that username into the proxied requests "Authorization: . . . " field. The proxied-to server can then verify that the request originated from the proxy server and extract and use that username as authenticated.

This functionality is described in the WASD_ROOT:[SRC.HTTPD]PROXYVERIFY.C module.

proxyMUNGE Utility

This utility (CGIplus script) can be used to rewrite HTTP response "Location:" fields, "Set-Cookie:" path and domain components and URLs in HTML and CSS content.

This functionality is described in the prologue to the code WASD_ROOT:[SRC.UTILS]PROXYMUNGE.C

Note

The proxyMUNGE Utility handles all response rewriting and so when employing it to perform reverse-proxy processing it is unnecessary to use the *proxy=reverse=location=<string>* mapping rule described in Redirection Location Field.

7.5.2 One-Shot Proxy

This looks a little like reverse proxy, providing access to a non-local resource via a standard (non-proxy) request. The difference allows the client to determine which remote resource is accessed. This works quite effectively for non-HTML resources (e.g. image, binary files, etc.) but non-self-referential links in HTML documents will generally be inaccessible to the client. This can provide provide scripts access to protocols they do not support, as with HTTP to FTP, HTTP to HTTP-over-SSL, etc.

Mappings appropriate to the protocols to be support must be made against the proxy service. Of course mapping rules may also be used to control whom or to what is connected.

```
[[the.proxy.service:port]]
# support "one-shot" non-proxy to proxy redirect
redirect  /http://*   http://*
redirect  /https://*  https://*
redirect  /ftp://*    ftp://*
# OK to process these (already, or now) proxy format requests
pass  http://*   http://*
pass  https://*  https://*
pass  ftp://*    ftp://*
```

The client may the provide the desired URL as the path of the request to the proxy service. Notice that the scheme provided in the desired URL can be any supported by the service and its mappings.

```
http://the.proxy.service:port/http://the.remote.host/path
http://the.proxy.service:port/https://the.remote.host/path
http://the.proxy.service:port/ftp://the.remote.host/pub/
```

7.5.3 DNS Wildcard Proxy

This relies on being able to manipulate host record in the DNS or local name resolution database. If a "*.the.proxy.host" DNS (CNAME) record is resolved it allows any host name ending in ".the.proxy.host" to be resolved to the corresponding IP address. Similarly (at least the Compaq TCP/IP Services) the local host database allows an alias like "another.host.name.proxy.host.name" for the proxy host name. Both of these would allow a browser to access "another.host.name.proxy.host.name" with it resolved to the proxy service. The request "Host:" field would contain "another.host.name.proxy.host.name".

Using this approach a fully functioning proxy may be implemented for the browser without actually configuring it for proxy access, where returned HTML documents contain links that are always correct with reference to the host used to request them. This allows the client an *ad hoc* proxy for selected requests. For a wildcard (CNAME) record the browser user may enter any host name prepended to the proxy service host name and port and have the request proxied to that host name. Entering the following URL into the browser location field

```
http://the.host.name.the.proxy.service:8080/path
```

would result in a standard HTTP proxy request for "/path" being made to "the.host.name:80". With the URL

```
https://the.host.name.the.proxy.service:8443/path
```

an SSL proxy request. Note that normally the well-known port would be used to connect to (80 for http: and 443 for https:). If the final, period-separated component of the wildcard host name is all digits it is interpreted as a specific port to connect to. The example

```
http://the.host.name.8001.the.proxy.service:8080/path
```

would connect to "the.host.name:8001", and

```
https://the.host.name.8443.the.proxy.service:8443/path
```

to "the.host.name:8443".

Note

It has been observed that some browsers insist that an all-digit host name element is a port number despite it being prefixed by a period not a colon. These browsers then attempt to contact the host/port directly. This obviously precludes using an all-digit element to indicate a target port number with these browsers.

This wildcard DNS entry approach is a more fully functional analogue to common proxy behaviour but is slightly less flexible in providing gatewaying between protocols and does require more care in configuration. It also relies on the contents of the request "Host:" field to provide mapping information (which generally is not a problem with modern browsers). The mappings must be performed in two parts, the first to handle the wildcard DNS entry, the second is the fairly standard rule(s) providing access for proxy processing.

```
[[the.proxy.service:port1]]
if (host:*.the.proxy.service:port1)
   redirect  *  /http://*
else
   pass  http://*   http://*
endif
```

The obvious difference between this and one-shot proxy is the desired host name is provided as part of the URL host, not part of the request path. This allows the browser to correctly resolve HTML links etc. It is less flexible because a different proxy service needs to be provided for each protocol mapping. Therefore, to allow HTTP to HTTP-over-SSL proxy gatewaying another service and mapping would be required.

```
[[the.proxy.service:port2]]
if (host:*.the.proxy.service:port2)
   redirect  *  /https://*
else
   pass  https://*   https://*
endif
```

7.5.4 Originating SSL

This proxy function allows standard HTTP clients to connect to Secure Sockets Layer (Chapter 4) services. This is very different to the CONNECT service (Section 7.3), allowing scripts and standard character-cell browsers supporting only HTTP to access secure services.

Standard username/password authentication is supported (as are all other standard HTTP request/response interactions). The use of X.509 client certificates (Section 4.3.12) to establish outgoing identity is not currently supported.

Enabling SSL

Unlike HTTP and FTP proxy it requires the service to be specifically configured using the [ServiceClientSSL] directive.

There are a number of Secure Sockets Layer related service parameters that should also be considered (see “WASD Web Services - Install and Config”). Although most have workable defaults unless [ServiceProxyClientSSLverifyCA] and [ServiceProxyClientSSLverifyCAfile] are specifically set the outgoing connection will be established without any checking of the remote server’s certificate. This means the host’s secure service could be considered unworthy of trust as the credentials have not been established.

```
[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceClientSSL]  enabled
```

7.6 Tunneling Using Proxy

WASD supports the CONNECT method which effectively allows tunneling of raw octets through the proxy server. This facility is most commonly used to allow secure SSL connections to be established with hosts on the ’other side’ of the proxy server. This basic mechanism is also used by WASD to provide an extended range of tunneling services. The term *raw* is used here to indicate an 8 bit, bidirectional, asynchronous exchange of octets between two entities, as a protocol family, not necessarily as an application (but can be so). Global proxy serving must be enabled (Section 7.1.1) and then each service must be configured and mapped according to the desired mode of tunneling. Disabling or setting timeouts appropriately on the mapped service is important if connections are not to be disrupted by general server timeouts on output and non-progress (quiescent connections).

7.6.1 [ServiceProxyTunnel] CONNECT

A service with this configuration is used as a target for CONNECT proxying (usually SSL through a firewall). The client expects an HTTP success (200) response once the remote connection is established, and HTTP error response if there is a problem, and once established just relays RAW octets through the proxy server (classic CONNECT behaviour).

```
# WASD_CONFIG_SERVICE
[[http://*:8080]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  connect

# WASD_CONFIG_MAP
[[*:8080]]
if (request-method:connect)
   pass *:443 *:443
   pass * "403 CONNECT only allowed to port 443."
endif
```

This configuration enables CONNECT processing and limits any connect to SSL tunneling (i.e. port 443 on the remote system).

7.6.2 [ServiceProxyTunnel] RAW

This allows any raw octet client (e.g. telnet) to connect to the port and by mapping be tunnelled to another host and port to connect to its service (e.g. a telnet service). The usual HTTP responses associated with CONNECT processing are not provided.

```
# WASD_CONFIG_SERVICE
[[http://*:10023]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  raw

# WASD_CONFIG_MAP
[[*:10023]]
if (request-method:connect)
   pass *:0 raw://another.host:23 timeout=none,none,none
endif
pass "403"
```

Telnet is used in the example above but the principle equally applies to any protocol that uses a raw 8 bit, bidirectional, asynchronous exchange of octets. Another example might be an SMTP service (port 25).

Chaining RAW

It is possible to have a raw tunnel establish itself through a proxy chain (Section 7.1.4) by transparently generating an intermediate CONNECT request to the up-stream proxy server. Note that not all CONNECT proxy will allow connection to just any specified port. For security reasons it it is quite common to restrict CONNECT to port 443.

```
# WASD_CONFIG_SERVICE
[[http://*:10025]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  raw

# WASD_CONFIG_MAP
[[*:10025]]
if (request-method:connect)
   pass *:0 raw://another.host:25 proxy=chain=proxy.host:8080
endif
pass "403"
```

Any error in connecting to the chained proxy, making the request, connecting to the destination, etc. (i.e. any error at all) is not reported. The network connection is just dropped. Use WATCH to establish the cause if necessary.

7.6.3 [ServiceProxyTunnel] FIREWALL

With this configuration a service expects that the first line of text from the client contains a host name (or IP address) and optional port (e.g. “the.host.name” or “the.host.name:23”). This allows a variable destination to be mapped. The usual HTTP responses associated with CONNECT processing are not provided.

```
# WASD_CONFIG_SERVICE
[[http://*:10023]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  FIREWALL

# WASD_CONFIG_MAP
[[*:10023]]
if (request-method:connect)
   pass *:* raw://*:23 timeout=none,none,none
   pass * raw://*:23 timeout=none,none,none
endif
pass "403"
```

The pass rules force the supplied domain name (and optional port) to be mapped to the telnet port (23). Of course the mapping rules could allow the supplied port to be mapped into the destination if desired.

Chaining FIREWALL

As with [ServiceProxyTunnel] RAW it is possible to chain FIREWALL services to an up-stream proxy server. See Chaining RAW.

7.6.4 Encrypted Tunnel

Up to this point the tunnels have merely been through the proxy server. It is possible to establish and maintain ENCRYPTED TUNNELS between WASD servers. SSL is used for this purpose. This is slightly more complex as both ends of the tunnel need to be configured.

```
                +------------+             +------------+
<-unencrypted->| WASD proxy |<-ENCRYPTED->| WASD proxy |<-unencrypted->
                +------------+             +------------+
```

This arrangement may be used for any stream-oriented, network protocol between two WASD systems. As it uses standard CONNECT requests (over SSL) it MAY also be possible to be configured between WASD and non-WASD servers.

The following example is going to maintain an encrypted tunnel between WASD servers running on systems KLAATU and GORT. It is designed to allow a user on KLAATU to connect to a specified port using a telnet client, and have a telnet session created on GORT, tunnelled between the two systems via an SSL encrypted connection.

Source of tunnel:

```
# KLAATU WASD_CONFIG_SERVICE
[[http://*:10023]]
[ServiceProxy]  enabled
[ServiceClientSSL]  ENABLED
[ServiceProxyTunnel]  RAW

# KLAATU WASD_CONFIG_MAP
[[*:10023]]
# if the client is on the local subnet
if (remote-addr:192.168.0.0/24 && request-method:connect)
   pass *:0 https://gort.domain:10443 timeout=none,none,none
endif
pass "403"
```

Destination of tunnel:

```
# GORT WASD_CONFIG_SERVICE
[[https://*:10443]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  CONNECT

# GORT WASD_CONFIG_MAP
[[*:10443]]
# limit the connection to a specific host
if (remote-addr:192.168.0.10 && request-method:connect)
   pass *:0 raw://gort.domain:23 timeout=none,none,none
endif
pass "403"
```

When a client connects to the service provided by port 10023 on system KLAATU the connection is immediately processed using a pseudo CONNECT request header. The service on this port is a proxy allowed to initiate SSL connections (client SSL). This service is mapped to system GORT port 10443, an SSL service that allows the CONNECT method (tunneling). KLAATU's proxy initiates an SSL connection with GORT. When established and the CONNECT request from KLAATU is received, it is mapped via a raw tunnel (8 bit, etc.) to its own system port 23 (the telnet service). Telnet is in use at both ends while encrypted by SSL inbetween! Note the use of network addresses and general fail rules used to control access to this service, as well as the disabling of timers that might otherwise shutdown the tunnel.

7.6.5 Encrypted Tunnel With Authentication

This arrangement is essentially a variation on example 4. It provides a cryptographic authentication of the originator (source) of the tunnel.

Source of tunnel:

```
# KLAATU WASD_CONFIG_SERVICE
[[http://*:10023]]
[ServiceProxy]  enabled
[ServiceClientSSL]  enabled
[ServiceProxyTunnel]  RAW
[ServiceClientSSLcert]  WASD_ROOT:[LOCAL]HTTPD.PEM

# KLAATU WASD_CONFIG_MAP
[[*:10023]]
# if the client is on the local subnet
if (remote-addr:192.168.0.0/24 && request-method:connect)
   pass *:0 https://gort.domain:10443 timeout=none,none,none
endif
pass "403"
```

Destination of tunnel:

```
# GORT WASD_CONFIG_SERVICE
[[https://*:10443]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  CONNECT
[ServiceProxyAuth]  PROXY

# GORT WASD_CONFIG_MAP
[[*:10443]]
# we'll be relying on X509 authentication
if (request-method:connect)
   pass *:0 raw://gort.domain:23 timeout=none,none,none
endif
pass "403"

# GORT WASD_CONFIG_AUTH
[[*:10443]]
[X509]
* r+w,param="[VF:OPTIONAL]",~4EAB3CBC735F8C7977EBB41D45737E37
```

This works by configuring the destination service to insist on proxy authorization. The authorization realm is X509 which causes the destination to demand a certificate from the source (Section 4.3.12). The fingerprint of this certificate is checked against the authorization rule before the connection is a allowed to procede.

7.6.6 Shared SSH Tunnel

The objective of this *raw* tunnel variant (see Section 7.6.2) is to allow tunneling of Secure Shell (SSH) via a client site proxy server CONNECT which is usually confined to port 443. Of course most Web servers are configured to provide SSL HTTP on port 443. Sharing of HTTP and SSH on the same port is a little problematic and involves some protocol detection. The following explanation of how it is implemented is so that the reader can understand the requirement for the "timeout quirk".

On configured services; WASD *peeks* at the incoming TCP byte stream to see if it's SSH protocol. If it is, the socket is associated with a proxy raw tunneling service and proxy tunneling initiated to a mapped SSH server. However (just to make it interesting) some SSH clients do not initiate their own exchange until after the SSH server, and so *peeking* only works for a subset of clients. Of course this is a Catch-22 of sorts! To provide for these clients; if an input timeout should occur (an SSH client waiting) WASD sets up the tunnel anyway and begins the proxy. The proxied SSH server should then initiate the protocol and the client respond. The directive [ServiceShareSSH] configured to be non-zero both enables this facility for a service and sets the input timeout period (which perhaps should be shorter than the default 30 seconds because such clients will wait that long for any SSH server response).

This approach seems to work well-enough in practice, although users need to be aware that some clients will pause (for the duration of the timeout period - the "timeout quirk") during initial connection setup.

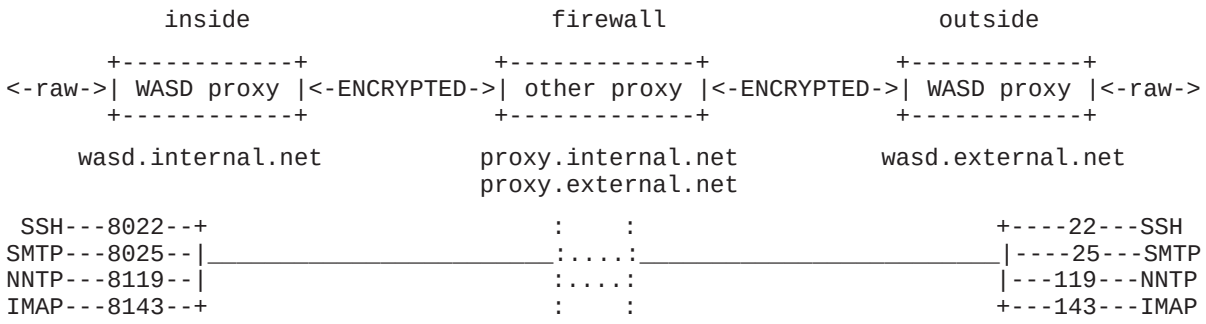
```
# WASD_CONFIG_SERVICE
[[https://*:443]
[ServiceShareSSH]  10

[[http://*:10022]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  raw

# WASD_CONFIG_MAP
[[*:443]
if (request-method:ssh)
   pass * raw://ssh.server.host:22 \
          service=the.proxy.host:10022 \
          timeout=none,none,none
endif

[[*:10022]]
pass "403"
```

This example shows an SSL service, the desired SSH service (which can be local or remote) and the internal proxy service that will provide the connection.

7.6.7 Complex Private Tunneling

When creating *raw* tunnels between WASD servers, and possibly in other circumstances, it is often useful to be able to signal *tunnel purpose* to the remote end. In this way a single destination port can support multiple tunneling purposes simply through mapping rules. An originating end can *inject* an HTTP request line, or full request, into the established tunnel connection, which can then be processed by the usual WASD request mapping, and from that alternate services provided based on the intent signalled by the originating end.

This somewhat complex but instructive example illustrates the potential utility and versatility of WASD tunneling. It involves an originating WASD server, a destination (service providing) WASD server, and just to make it interesting an intermediate chained HTTP proxy server (not WASD). The idea is to provide access to various application services not necessarily supported by intermediate HTTP proxies and/or gateways. Four services will be supported by the example; SSH, NNTP IMAP and SMTP.



Internal Services

These are the services assigned on the WASD server on the inside of the proxy/gateway. Note that there is one per application to be tunneled. For simplicity each service port number has been selected to parallel the well-known application port number. Note that *proxy* is enabled on each (allowing them to initiate outgoing connections) and each has *SSL* enabled (further allowing them to initiate encrypted connections).

```
# client SSH
[[http://*:8022]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  RAW
[ServiceClientSSL]  enabled

# client SMTP
[[http://*:8025]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  RAW
[ServiceClientSSL]  enabled

# client IMAP
[[http://*:8143]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  RAW
[ServiceClientSSL]  enabled

# client NNTP
[[http://*:8119]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  RAW
[ServiceClientSSL]  enabled
```

Each client application (i.e. IMAP, SSH) must be configured to connect to its corresponding service port (e.g. IMAP to 8143, SMTP to 8025).

Internal Mapping

These mappings are made on the WASD server on the inside of the proxy/gateway. The rules essentially initiate an outgoing encrypted (SSL) connection to the host *wasd.external.net* supporting the external WASD proxy server. Each is also configured not to connect directly but to request the chained proxy server *proxy.internal.net* to establish the connection on their behalf.

```
!##### SSH #####
[[*:8022]]
pass * https://wasd.external.net:443 notimeout \
proxy=tunnel=request="CONNECT wasd-ssh" \
proxy=chain=proxy.internal.net:8080

!##### SMTP #####
[[*:8025]]
pass * https://wasd.external.net:443 \
proxy=tunnel=request="CONNECT external-smtp" \
proxy=chain=proxy.internal.net:8080

!##### NNTP #####
[[*:8119]]
pass * https://wasd.external.net:443 \
proxy=tunnel=request="CONNECT external-nntp" \
proxy=chain=proxy.internal.net:8080

!##### IMAP #####
[[*:8143]]
pass * https://wasd.external.net:443 \
proxy=tunnel=request="CONNECT external-imap" \
proxy=chain=proxy.internal.net:8080
```

If the up-stream proxy server successfully connects to *wasd.external.net* port 443 the proxy server allows the byte-stream to be asynchonously and bidirectionally exchanged with the internal WASD server outgoing connection. This internal WASD server has initiated an SSL connection and the external server port 443 expects SSL so they can now both negotiate an SSL-encrypted channel essentially directly with each other.

External Services

The external WASD service configuration is very simple, a single SSL port.

```
# general SSL service
[[https://wasd.external.net:443]]

# outgoing proxy/tunnel service
[[http://wasd.external.net:1234]]
[ServiceProxy]  enabled
[ServiceProxyTunnel]  raw
[ServiceClientSSL]  ENABLED
```

Connections to the 443 port are expected to undertake an SSL negotiation to establish an encrypted channel. This includes incoming tunnel connections. The service on port 1234 is required to support the connections outgoing from the external WASD server to the application server ports.

External Mapping

These mappings are all applied to requests at port 443 on the external WASD server *wasd.external.net*. Each rule checks three request characterstics. First, the request method, "CONNECT". Second, the request URI, varies according to the request. These are the request data injected by the internal WASD server *wasd.internal.net* using the *set=proxy=tunnel=request=* mapping rule on the outgoing connection. Third, the originating host (*proxy.external.net*) address adds an extra filter on from where this facility may be used. The respective *pass* of the matching rule then initiates an outgoing connection to the

respective application server's well-known port. A timeout is applied to limit connection times.

```
!# SSH tunneling
[[*:443]]
if (request-method:CONNECT && \
request-uri:"wasd-ssh" && \
remote-addr:205.3.*) \
pass * raw://wasd.external.net:22 service=*:1234 timeout=noprogress=00:00:50

!# SMTP tunneling
[[*:443]]
if (request-method:CONNECT && \
request-uri:"external-smtp" && \
remote-addr:205.3.*) \
pass * raw://smtp.isp.net:25 service=*:1234 timeout=noprogress=00:00:50

!# NNTP tunneling
[[*:443]]
if (request-method:CONNECT && \
request-uri:"external-nntp" && \
remote-addr:205.3.*) \
pass * raw://news.isp.net:119 service=*:1234 timeout=noprogress=00:00:*

!# IMAP tunneling
[[*:443]]
if (request-method:CONNECT && \
request-uri:"external-imap" && \
remote-addr:205.3.*) \
pass * raw://imap.isp.net:143 service=*:1234 timeout=noprogress=00:00:50

!# disable general 1234 service usage
[[*:1234]]
pass * 403 "Internal use only!"
```

Example In Action

Now let's look at an actual example usage. Consider the internal user's IMAP application, say Thunderbird, is configured to use an IMAP server at host *wasd.internal.net* port 8143. The internal user activates Thunderbird which then intiates an TCP/IP connection to the configured IMAP server expecting to commence the IMAP application protocol.

This connection arrives at *wasd.internal.net* port 8143 which has a WASD *raw* tunnel service listening. The connection is accepted and request processing commences. Mapping rules applied to port 8143 initiate an SSL connection to host *wasd.external.net* which is not directly accessable because of the firewall and must be connected to using the HTTP proxy server *proxy.internal.net* as an intermediary. This is specified in the same mapping rule. The mapping rule also injects an HTTP request header providing request characteristics that can be identified and acted upon by the external server.

The internal WASD server initiates a connection to the proxy server *proxy.internal.net* acting as part of the firewall. As it is endeavouring to initiate an SSL connection with the external *wasd.external.net* host this proxy connection uses a CONNECT request specifying *wasd.external.net* port 443. The proxy server establishes a connection with the host *wasd.external.net* at port 443. Once the connection is established it becomes an asynchronous, bidirectional channel between *wasd.internal.net* and *wasd.external.net* with the proxy server as a conduit.

The service connection just established is expecting an SSL negotiation in an attempt to establish an encrypted channel. When this negotiation concludes successfully the communications between *wasd.internal.net* and *wasd.external.net* become opaque to all external listeners including *proxy.internal.net*.

The encrypted connection now established, the request begins to be processed by the WASD server at *wasd.external.net*. A number of mapping rules apply to port 443. Each rule compares the injected request method and URI until, in this case, the *external-imap* rule matches. This rule specifies that a raw connection be established with the host *imap.isp.net* at port 143 using the proxy-capable port 1234 service. A timeout limits the duration this connection can be held unused.

The IMAP application server at *imap.isp.external* port 143 accepts the connection at begins to communicate using the IMAP protocol.

There is now a raw (8 bit, asynchronous, bidirectional) connection from the Thunderbird client to *wasd.internal.net*, (encrypted) through to *proxy.internal.net*, (encrypted) through to *wasd.external.net*, and raw to the IMAP server at *imap.isp.net*. This raw connection will be used for communication between Thunderbird and the IMAP server using the IMAP application protocol.

7.7 Browser Proxy Configuration

The browser needs to be configured to access URLs via the proxy server. This is done using two basic approaches, manual and automatic.

7.7.1 Manual

Most browsers allow the configuration for access via a proxy server. This commonly consists of an entry for each of the common Web protocol schemes (“http:”, “ftp:”, “gopher:”, etc.). Supply the configured WASD proxy service host name and port for the HTTP scheme. This is currently the only one available. This would be similar to the following example:

```
http: www.example.com 8080
```

To exclude local hosts, and other servers that do not require proxy access, there is usually a field that allows a list of hosts and/or domain names for which the browser should not use proxy access. This might be something like:

```
www.example.com,example.com,example.com
```

7.7.2 Automatic

At least Netscape Navigator/Communicator and Microsoft Internet Explorer (4.*n* and following) provide the facility to download a small JavaScript function for establishing proxy policy. Information on this function and its deployment may be found at

http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html

The following is a very simple proxy configuration JavaScript function. This specifies that all URL host names that aren’t full qualified, or that are in the “example.com” domain will be connected to directly, with all other being accessed via the specified proxy server.

```
function FindProxyForURL(url,host)
{
   if (isPlainHostName(host) ||
       dnsDomainIs(host, ".example.com"))
      return "DIRECT";
   else
      return "PROXY www.example.com:8080; DIRECT";
}
```

This JavaScript is contained in a file with a specific, associated MIME file type, "application/x-ns-proxy-autoconfig". For WASD it is recommended the file be placed in WASD_ROOT:[LOCAL] and have a file extension of .PAC (which follows Netscape naming convention).

The following WASD_CONFIG_GLOBAL directive would map the file extension to the required MIME type:

```
[AddType]
.PAC  application/x-ns-proxy-autoconfig  -  proxy autoconfig
```

This file is commonly made the default document available from the proxy service. The following example shows the HTTP$MAP rules required to do this:

```
[www.example.com:8080]
pass http://* http://*
pass / /wasd_root/local/proxy.pac
pass *
```

All that remains is to provide the browser with the location from which load this *automatic proxy configuration* file. In the case of the above set-up this would be:

```
http://www.example.com:8080/
```

A template for a proxy auto-configuration file may be found at WASD_ROOT:[EXAMPLE]PROXY_AUTOCONFIG.TXT.

Chapter 8
Instances and Environments

WASD *instances* and *environments* are two distinct mechanisms for supporting multiple WASD server processes on a single system.

Server instances are multiple, cooperating server processes providing the same set of configured resources.

Server environments are multiple, independent server processes providing differently configured resources.

8.1 Server Instances

The term *instance* is used by WASD to describe an autonomous server process. WASD will support multiple server processes running on a single system, alone or in combination with multiple server processes running across a cluster. This is not the same as supporting multiple virtual servers (see "WASD Web Services - Install and Config") When multiple instances are configured on a single system they cooperate to distribute the request load between themselves and share certain essential resources such as accounting and authorization information.

WARNING

Versions earlier than Compaq TCP/IP Services v5.3 and some TCPware v5.*n* (at least) have a problem with socket listen queuing that can cause services to "hang" (should this happen just disable instances and restart the server). Ensure you have the requisite version/ECO/patch installed before activating multiple instances on production systems!

8.1.1 VMS Clustering Comparison

The approach WASD has used in providing multiple instance serving may be compared in many ways to VMS clustering.

A cluster is often described as a loosely-coupled, distributed operating environment where autonomous processors can join, process and leave (even fail) independently, participating in a single management domain and communicating with one another for the purposes of resource sharing and high availability.

Similarly WASD instances run in autonomous, detached processes (across one or more systems in a cluster) using a common configuration and management interface, aware of the presence and activity of other instances (via the Distributed Lock Manager and shared memory), sharing processing load and providing rolling restart and automatic "fail-through" as required.

Load Sharing

On a multi-CPU system there are performance advantages to having processing available for scheduling on each. WASD employs AST (I/O) based processing and was not originally designed to support VMS kernel threading. Benchmarking has shown this to be quite fast and efficient even when compared to a kernel-threaded server (OSU) across 2 CPUs. The advantage of multiple CPUs for a single multi-threaded server also diminishes where a site frequently activates scripts for processing. These of course (potentially) require a CPU each for processing. Where a system has many CPUs (and to a lesser extent with only two and few script activations) WASD's single-process, AST-driven design would scale more poorly. Running multiple WASD instances addresses this.

Of course load sharing is not the only advantage to multiple instances . . .

Restart

When multiple WASD instances are executing on a node and a restart is initiated only one process shuts down at a time. Others remain available for requests until the one restarting is again fully ready to process them itself, at which point the next commences restart. This has been termed a *rolling restart*. Such behaviour allows server reconfiguration on a busy site without even a small loss of availability.

Fail-Through

When multiple instances are executing on a node and one of these exits for some reason (resource exhaustion, bugcheck, etc.) the other(s) will continue to process requests. Of course requests in-progress by the particular instance at the time of instance failure are disconnected (this contrasts with the rolling restart behaviour described above). If the former process has actually exited (in contrast to just the image) a new server process will automatically be created after a few seconds.

The term *fail-through* is used rather than *failover* because one server does not commence processing as another ceases. All servers are constantly active with those remaining immediately and automatically taking all requests in the absence any one (or more) of them.

8.1.2 Considerations

Of course "there is no such thing as a free lunch" and supporting multiple instances is no exception to this rule. To coordinate activity between and access to shared resources, multiple instances use low-level mutexes and the VMS Distributed Lock Manager (DLM). This does add some system overhead and a little latency to request processing, however as the benchmarks indicate increases in overall request throughput on a multi-CPU system easily offset these costs. On single CPU systems the advantages of rolling restart and fail-through need to be

assessed against the small cost on a per-site basis. It is to be expected many low activity sites will not require multiple instances to be active at all.

When managing multiple instances on a single node it is important to consider each process will receive a request in round-robin distribution and that this needs to be considered when debugging scripts, using the Server Administration page and the likes of WATCH, etc. (see Section 10.1).

8.1.3 Configuration

If not explicitly configured only one instance is created. The configuration directive [Instance-Max] allows multiple instances to be specified (see “WASD Web Services - Install and Config”) When this is set to an integer that many instances are created and maintained. If set to “CPU” then one instance per system CPU is created. If set to “CPU-*integer*” then one instance for all but one CPU is created, etc. The current limit on instances is eight, although this is somewhat arbitrary. As with all requests, Server Administration page access is automatically shared between instances. There are occasions when consistent access to a single instance is desirable. This is provided via an *admin service* (see “WASD Web Services - Install and Config”)

When executing, the server process name appends the instance number to the “WASD”. Associated scripting processes are named accordingly. This example shows such a system:

```
   Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
21600801 SWAPPER         HIB     16        0   0 00:06:53.65         0      0
21600807 CLUSTER_SERVER  HIB     12     1879   0 00:01:14.51        91    112
21600808 CONFIGURE       HIB     10       30   0 00:00:01.46        47     23
 . . .
21600816 ACME_SERVER     HIB     10    71525   0 00:01:28.08       508    713 M
21600818 SMISERVER       HIB      9    11197   0 00:00:02.29       158    231
21600819 TP_SERVER       HIB      9  1337711   0 00:05:55.78        80    105
 . . .
216421F1 WASD1:80        HIB      5  5365731   0 00:23:12.86     37182   7912
2164523F WASD2:80        HIB      5  5347938   0 00:23:31.41     38983   7831
2162BA5D WASD_WOTSUP     HIB      3     2111   0 00:00:00.47       735    518
2164ABCF WASD1:80-651    LEF      6    57884   0 00:00:16.71      3562   3417
2164CBDB WASD2:80-612    LEF      4    19249   0 00:00:04.16      3153   3116
21631BDC WASD2:80-613    LEF      5    18663   0 00:00:07.19      3745   3636
2164BBE6 WASD1:80-658    LEF      5     3009   0 00:00:00.94      2359   2263
 . . .
```

8.2 Server Environments

WASD server environments allow multiple, distinctly configured environments to execute on a single system. Generally, WASD’s unlimited virtual servers and multiple account scripting eliminates the need for multiple execution environments to kludge these requirements. However there may be circumstances that make this desirable; regression and forward-compatibility testing comes to mind.

First some general comments on the design of WASD.

- WASD creates and populates it’s own logical name table (see “WASD Web Services - Install and Config”).

It also adds at least the WASD_FILE_DEV[*n*] logical name to the SYSTEM logical name table.

- WASD creates and uses rights identifiers.

  Installation creates and associates specific rights identifiers with separate accounts for server and script execution. Some specifically named identifiers have functional meaning to the server. Server startup can create and associate rights identifers used to manage the server run-time environment.

- WASD makes extensive use of the DLM to coordinate WASD activities system- and cluster-wide.

  All executing server images are aware of all other executing server images on the same system and within the same cluster. This performs all manner of coordination (e.g. instance recovery, instantiated services) and data exchange (e.g. $HTTPD/DO=MAP/ALL) activities.

- WASD uses global sections to accumulate data and for communication between WASD instances.

  Some of these are by default permanent and remain on a system unless explicitly removed.

- WASD uses detached scripting processes.

  As it's possible to $STOP a server process (and thereby prevent it's run-down handlers from cleaning up those detached processes). It therefore needs to be able to recognise as its 'own' and clean any such 'orphaned' processes up next time it starts. It does this by having a rights identifier associated with the server process name (e.g. WASD:80 grants its scripting processes WASD_PRC_WASD_80, a second instance WASD2:80, WASD_PRC_WASD2_80, etc.)

All of these mechanisms support multiple, independent environments on a single system. Due to design and implementation considerations there are fifteen such environments available per system. The primary (default) is one. Environments two to fifteen are available for site usage. (Demonstration mode, /DEMO uses environment zero.) Server *instances* (Section 8.1) share a single environment.

There are two approaches to provisioning such multiple, independent environments.

8.2.1 Ad Hoc Server Wrapper

This is a DCL procedure that allows virtually any WASD release HTTP server to be executed in a detached process, either by itself or concurrently with a full release or other ad hoc detached server. The server image and associated configuration files used by this process can be specified within the procedure allowing completely independent versions and environments to be fully supported.

Full usage instructions may be found in the example procedure(s) in WASD_ROOT:[EXAMPLE]*ADHOC*.CO

Two versions are provided, one for pre-v10 and one for post-v10 (due to changes in logical naming schema).

8.2.2 Formal Environments

Although the basic infrastructure for supporting multiple environments (i.e. the 0..15 environment number) has been in place since version 8, formal support in server CLI qualifiers and DCL procedures has only been available since version 10. To support version 9 or earlier environments the Section 8.2.1 must be used.

WASD version 10 startup and other run-time procedures have been modified to support running multiple WASD environments simply from independent WASD file-system trees. The standard STARTUP.COM procedure accepts the WASD_ENV parameter to specify which environment (1..15) the server should execute within (primary/default is 1). The procedure then derives the WASD_ROOT logical name from the location of the startup procedure.

For example:

```
$! start current release
$ WASD_STARTUP = "/SYSUAF=(ID,SSL)/PERSONA"
$ @DKA0:[WASD_ROOT.STARTUP]STARTUP.COM
$! start previous release in environment 2
$ WASD_ENV = 2
$ @DKA0:[WASD_ROOT_MINUS1.STARTUP]STARTUP.COM
```

8.2.3 Considerations

WASD environments each fully support all WASD features and facilities (including multiple server instances) with the exception of DECnet scripting where because of DECnet objects' global (per-system) definition only the one must be shared between environments.

Per-environment configuration must be done in its own WASD_ROOT part of the file-system and logical names must be defined in the environment's associated logical name table. The site administrator must keep track of which environment requires to be accessed from the command-line and set the process logical name search list using the appropriate

```
$ @WASD_FILE_DEV[n]
```

where *n* can be a non-primary environment number (see "WASD Web Services - Install and Config").

It is not possible to have multiple environments bind their services to the same IP address and port (for fundamental networking reasons). Unless the network interface is specifically multi-homed for the purpose, services provided by separate environments must be configured to use unique IP ports.

Non-primary environments (2 . . . 15) prefix the environment as a (hex) digit before the "WASD" in the process name. The above example when executing, each with a single scripting process, would appear in the system as (second environment providing a service on port 2280):

```
  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
00000101 SWAPPER         HIB     16        0   0 00:00:11.98         0      0
 . . .
00000111 ACME_SERVER     HIB     10     6247   0 00:00:12.63       540    611 M
00000112 QUEUE_MANAGER   HIB     10      328   0 00:00:00.18       136    175
00000122 TCPIP$INETACP   HIB     10  1249419   0 00:07:33.95       401    326
00000123 TCPIP$ROUTED    LEF      6  3495839   0 00:01:15.49       166    165 S
 . . .
00000468 WASD:80         HIB      6   132924   0 00:01:29.26     17868   2856
0000046D 2WASD:2280      HIB      6   129344   0 00:01:29.26     17712   2840
0000049D WASD:80-8       LEF      4     4449   0 00:00:00.67       934    194
00000503 2WASD:2280-2    LEF      4      565   0 00:00:00.28       732    102
 . . .
```

Cleaning Up

As described earlier each environment creates and maintains logical name table(s) and system-level name(s), detached scripting processes, lock resources and permananent global sections. Lock resources disappear with the server processes. Logical names, global sections, rights identifiers and occasionally detached scripting processes may require some cleaning up when a non-primary environment's use is concluded.

Chapter 9
Server Administration

The online Server Administration facility provides a rich collection of functionality, including server control, reports and configuration. Some of these are intended as general administration tools while other provide more detailed information intended for server debugging and development purposes.

online graphic

The value of the WATCH facility Chapter 10 as a general configuration and problem-solving tool cannot be overstated.

All server configuration files, with the execption of the authentication databases, are plain text and may be modified with any prefered editor. However the majority of these can also be administered online through a browser. In addition the *update* facility allows some administration of file system portions of the Web. See Chapter 12.

Access to many portions of the package is constrained by file protections and directory listing access files. See Section 3.10.9 for a method for circumventing these restrictions.

9.1 Access Before Configuration

It is often a significant advantage for the inexperienced administrator on a new and largely unconfigured installation to be able to gain access to the facilities offered by Server Administration, particularly the WATCH facility (Chapter 10). This can be done quite simply by using the authentication skeleton-key (Section 3.12). This allows the site administrator to register a username and password from the command-line that can be used to gain access to the server. In addition, the server ensures that requesting an otherwise non-authorized Server Administration facility generates a challenge which invokes a username/password dialog at the browser allowing the user to enter the previously registered username and password and gain access.

Method

- Register the skeleton-key username and password.

```
$ HTTPD == "$WASD_EXE:HTTPD_SSL.EXE"
$! HTTPD == "$WASD_EXE:HTTPD.EXE"
$ HTTPD /DO=AUTH=SKELKEY=_username:password
```

  Note that the username must begin with an underscore, be at least 6 characters, is delimited by a colon, and that the password must be at least 8 characters. By default this username and password remains valid for 60 minutes. **Choose strings that are less-than-obvious!**

- Access the server via a browser and use the server Server Administration facility.

```
http://the.host.name:port/httpd/-/admin/
```

- After use the skeleton-key may be explicitly cancelled if desired.

```
$ HTTPD /DO=AUTH=SKELKEY=0
```

9.2 Access Configuration

One established the site should make the Server Administration facility a configured facility of the site. The value of its facilities cannot be overstated.

It is also recommended that for production sites the path to these reports be controlled via authentication and authorization, using both host and username restrictions, similar to the following:

```
[WHATEVER-REALM]
/httpd/-/admin/*  host.ip.addr,~WebMaster,~WhoEverElse,r+w
```

If a full authorization environment is not required but administration via browser is still desired restrict access to browsers executing on the server system itself, using an appropriate SYSUAF-authenticated username. Provision of a VMS account for server administration only is quite feasable, see Section 3.10.6.

```
[VMS]
/httpd/-/admin/*  #localhost,~username,r+w
```

If SSL is in use (Chapter 4) then username/password privacy is inherently secured via the encrypted communications. To restrict server administration functions to this secure environment add the following to the WASD_CONFIG_MAP configuration file:

```
/httpd/-/admin/*  "403 Access denied."  ![sc:https]
```

When using the *revise* capability of the Server Administration facility is necessary to comply with all the requirements for Web update of files. This is discussed in general terms in Chapter 12. Revision of server configuration files requires path permissions allowing write access for the username(s) doing the administration, as well as the required ACL on the target directory (in the following example WASD_ROOT:[LOCAL]).

```
[VMS]
/httpd/-/admin/*  #localhost,~username,r+w
/wasd_root/local/*  #localhost,~username,r+w
```

It is possible to allow general access to the Server Administration facility and reports while restricting the ability to initiate server actions such as a restart! Using the WORLD realm against the path is necessary, for the obvious security reason, the server administration module will not allow itself to be used without an authenticated username, provided as a pseudo-authenticated "WORLD".

```
[VMS]
/httpd/-/admin/control/*  #localhost,~username,r+w
[WORLD]
/httpd/-/admin/* r
```

When GZIP compression is configured for the server (see "WASD Web Services - Install and Config" document, "GZIP Encoding" section) it is not by default applied to Server Admin reports or other pages. It can be applied, selectively if desired, using mapping rules. For instance, to apply it to all requests not from the local intranet a rule similar to the following can be added before the Server Admin path mapping itself.

```
if (!remote-addr:192.168.0.0/8) set /httpd/-/admin/* response=GZIP=all
pass /httpd/-/admin/* /httpd/-/admin/*
```

GZIP content-encoding can never be applied to WATCH reports.

9.3 Server Instances

With a single instance (see Section 8.1) access to Server Administration reports, etc. is always serviced by the one server process. If multiple instances are configured then in common with all requests administration requests will be serviced by any one of the associated processes depending on the momentary state of the round-robin distribution.

There are many circumstances where it is preferable to access only the one server. This can be accomplished for two differing objectives.

1. To facilitate access to a specific instance's Server Administration page, including instance-specific reports etc. This is provided through the use of an *administration service* port (see "WASD Web Services - Install and Config") available from the Server Administration page.
2. The Server Administration page (Control Section) and the command-line (Section 9.7.9) provides the capability to explicitly set the number of instances supported, overriding any configuration directive. After explicitly setting this using either means the server must be restarted. The explicit startup setting remains in effect until it's changed to "max" allowing the WASD_CONFIG_GLOBAL configuration directive [InstanceMax] to once again determine the number of instances required.

The latter approach is particularly useful when performing detailed WATCH activities (Chapter 10).

When multiple per-node instances are executing the Server Administration pages and reports all include an indication of which process serviced the request. When accessing no instance in particular the process name is presented in parentheses after the page title

```
HTTPd www.example.com:80
Server Administration  (HTTPd:80)
```

When a particular instance's administration service port is being used the process name is separated from the page title by a hyphen

```
HTTPd www.example.com:80
Server Administration - HTTPd:80
```

9.4 HTTPd Server Reports

The server provides a number of internally generated reports. Some of these are of general interest. Others are more for evaluating WASD behaviour and performance for development purposes. Appropriate reports have a refresh selector allowing the report to be updated at the selected period. The following list is in the approximate order in which they occur top-to-bottom, left-to-right in the menu layout.

It is possible to use this facility standalone, without configuring authorization (Section 9.1).

- **Statistics -** Server process up-time, CPU-time and other resources consumed, number of connections processed, number of requests of each HTTP method, type of processing involved (HTTPd module used), number of bytes processed, etc.
- **Log -** Display the server process (SYS$OUTPUT) log.
- **Configuration -** A tabular summary of the server's current configuration. This is a convenient method for viewing the information from the WASD_CONFIG_GLOBAL file.
- **Services -** A tabular report listing the current services (virtual servers) and the service-specific parameters.
- **Messages -** A tabular report of the server's current message database, multiple languages shown if configured that way.
- **Mapping -** All loaded mapping rules and any cached USER rule paths. A selector allows rules applying only to one particular virtual server to be displayed.
- **Path Authorization -** If authorization is in use (Chapter 3) this report lists the paths with associated authorization and access control.
- **User Authentication -** List any users that have been authorized since the server was last started, the realm authorized from, the group it applies to (if any), and what the user's capabilities are (allowed HTTP methods). A time-stamp and counters provide additional information.
- **Secure Sockets -** The SSL report lists counts of the number of SSL transactions initiated and completed, along with session cache statistics for the currently connected SSL service. It also lists the ciphers available and current session information. Other reports allow the Certificate Authority (CA) database to be view and edited, if available due to X.509 authentication being enabled.
- **AlnFlt -** On Alpha and Itanium, memory access alignment faults are constantly monitored. This displays the accumulated statistics since the most recent startup. Should always be zero!

- **Cache -** Allows monitoring of cache behaviour and performance, as well as the files currently in the cache (see “WASD Web Services - Install and Config”).
- **Cluster -** For clustered systems generates a report similar to the *System Report* but with a cluster emphasis.
- **DCL Scripting -** Provides some DCL, CGI and CGIplus scripting information.

  DCL module statistics (same information as displayed in the server statistics report). These are cumulative for the entire life of the system (unless zeroed).

  Process information shows how many actual processes exist at the time of the report, as indicated by the PID and bolded, non-zero liftime (in minutes). The *soft-limit* specifies how many CGIplus scripts are allowed to continue existing before the least used is deleted and the *hard-limit* show how many processes may actually exist at any one time (the margin allows for process deletion latency). A count of how many times the CGIplus processes have been explicitly purged (button available on this report page). The *life-time* of zombie processes (in minutes, zero implying use of zombies is disabled) and the number that have been purged due to expiry. CGIplus process life-time (in minutes, zero implying indefinite), the number purged due to life-time expiry and the number of CGIplus processes that the server has actually purged (deleted) to maintain the soft-limit margin specified above.

  Each of the allocated process data structures is listed. There may be zero up to hard-limit items listed here depending on demand for DCL activities and the life of the server. Items with a PID shown indicate an actual process existing. This can be a zombie process or a CGIplus process. If no process is indicated then the other information represents the state the last time the item’s associated process completed. Information includes the script (URL-style path) or DCL command, total count of times the item has been used and the last time it was. The zombie count indicates the number of time the same process finished a request and entered the *zombie* state. The CGIplus column indicates it is/was a CGIplus script and shows the total number of times that particular script has been/was used. If the process is currently in use the client information show the client host name.

  If any processes are associated with any data structure a *purge* button is provided that forces all processes to be deleted. This can be useful if a new script image is compiled and it is required all scripts now use this. If a script is currently processing a request the process deletion occurs when that processing is complete. The purge button **does not force** a process to delete, so a second button **forces** all processes to delete immediately. This can be used to forceably clear errant scripts, etc., but be warned script processing is indiscrimately stopped!
- **DECnet Scripting -** DECnet module information shows totals for DECnet scripting usage and the DECnet connection list.

  This list will grow, up to the specified configuration maximum, as conconurrent scripting demand occurs. Maintained connections are indicated by the bolded, non-zero lifetime (in minutes). When this reaches zero the task is disconnected. The current/last task for that connection is indicated, along with the number of times the connection was reused and a total number of uses for that list item.

  Purge and *force* buttons allow current links to be broken after request completion or forcibly disconnected.

- **HTTP -** Reports HTTP/2 and HTTP/1.*n* statistics together as well as providing a list of current HTTP/2 connections with some per-connection data. See Chapter 5 for details.
- **Lock -** Lists the names and status of all lock resources used to manage single and multiple instances across single systems or a cluster. This report is more relevant for evaluating and debugging WASD behaviour.
- **Match -** To assist with the refinement of string matching patterns (see "WASD Web Services - Install and Config"). this report allows the input of target and match strings and allows direct access to the server's wildcard and regular expression matching routines. Successful matches show the matching elements and a substitution field allows resultant strings to be assessed.
- **Memory -** Provides a report and does an integrity check on each of the Virtual Memory (VM) zones employed by the WASD HTTPd.
- **Process -** Lists all processes on the current system owned by the server account. From this list a process can be selected to have a "SHOW PROCESS /ALL" performed on it, displayed on a report page.
- **Proxy -** If proxy serving is enabled a report providing statistics on the various HTTP methods used, network and cache traffic, cache reads and writes, requests not cachable, and host name lookup are provided. This may used to help guage the effectiveness of the cache.
- **Request -** Lists in-progress requests (always shows at least your own connection accessing this report :-) Additional buttons after the report allow selection of a report that in addition displays current persistent network connections, requests currently under throttle control, and if enabled a list (history) of the most recent requests (enabled by the configuration parameter [RequestHistory]). Current requests may be selected for *one-shot* WATCH-processing reports from this page (Chapter 10).

  Two other diagnostic tools are available from the same link. The first, *WATCH-peek Report*, providing a snapshot of the contents selected internal fields and data structures of the request. This is primarily intended as a problem investiagtion and development tool, and will be of limited value without an understanding of server internals. The second accesses the "peek" internals plus a one-shot WATCH-processing report.

  For servers handling a great quantity of concurrent traffic this can generate a very large report. The *Supervisor* report can also provide a profile of the servers current load.
- **Supervisor -** Provides a simple table displaying each timer list and any associated request count. Shows how many requests are set be scanned and evaluated for continued processing every so-many seconds. For very busy servers this is another method for gaining an idea of the traffic profile (this is perhaps more meaningful for those with an understanding of WASD internals).
- **System -** Shows the system, all users, memory and CPU status as a single report.
- **Throttle -** This report provides a list of paths with throttle rules mapped against them. It provides the throttle values along with current and history activity counters.
- **WATCH -** This report provides an online, real-time, in-browser-window view of request processing on the **running server**. See Chapter 10 for details.

- **WebDAV -** Provides configuration and statistics.
- **WebSocket -** Lists in-progress WebSocket requests with connection statistics and the scripting process associated with.
- **Activity -** Provide a graphical *snapshot* of server activity of a given period.

  The statistics are stored in a permanent global section and so carry-over between server restarts. Where multiple instances are executing the data represents an accumulation of all instances' processing. It is enabled by the configuration parameter [ActivityDays]. The Server Administration facility provides several, represented as a period of hours before the present time. Number of requests and bytes sent to the client are represented by a histogram with respective means for each by a line graph. A bar across the column of the request histogram indicates the peak number of concurrent requests during the period. A *greyed* area indicates no data available for that time (i.e. before the latest server startup, or in the future).

  Server startup and shutdown events are indicated by solid, vertical lines the full height of the graph (see example for a restart event).

  startup - green
  shutdown - black
  restart - grey
  error exit - red

  Activity data is accumulated on a per-minute basis. This is the maximum granularity of any report. When reports are selected that can display less than this one minute granularity (i.e. with periods greater than four hours) the value shown is the **peak** of the number of minutes sampled for display. This better represents the load on the server than would a mean of those samples.

  The graph is an image map, various regions of which allow the selection of other reports with different periods or durations. This allows previous periods to be examined at various levels of detail using the graph for navigation. Various sections may have no mapping as appropriate to the current report.

  For multiple hour reports the upper and lower sections have distinct functions. The middle 50% of the upper section allows the same end time (most commonly the current hour) to be examined over twice the current period, in this case it would be over eight hours. The left 25% allows the previous fours hours to be viewed (if such data exists), and for non-current reports the right 25% allows the next four hours to be viewed. The lower half can be divided into sections representing hours or days depending on the period of the current report. This allows that period to be viewed in greater detail. For single hour reports this section, of course, is not mapped.

  Remember that the URL of the mapped section will be displayed in the status bar of the browser. As the URL contains time components it is not a difficult task to decipher the URL displayed to see the exact time and period being selected.

9.5 HTTPd Server Revise

The server provides a comprehensive configuration revision facility.

- **Configuration -** A form-driven interface allows the current configuration of the server to be altered online. This configuration may then be saved to the on-disk file and then the server could be restarted using the new parameters. The source of the current configuration can be either the server itself (from its volatile, in-memory parameters) or from the on-disk configuration file. In addition it is possible to directly edit and update the on-disk file.
- **Services -** A form-driven interface allows service (virtual server) configuration. It is also possible to directly edit and update the on-disk file. The server must be restarted for service changes to take effect.
- **Messages -** A form-driven interface allows the the server messages to be modified. It is also possible to directly edit and update the on-disk file. The server can then be restarted to use the modified database (Section 9.6).
- **Mapping -** No form-driven interface is currently available for changing the mapping rules. However it is possible to directly edit and update the on-disk file. The mapping rules could then be reloaded, changing the current server rules (Section 9.6).
- **Path Authorization -** No form-driven interface is currently available for changing the path authorization configuration. However it is possible to directly edit and update the on-disk file. The path authorization directives could the be reloaded, changing the current server authorization (Section 9.6).
- **User Authentication -** User authentication comprises a number of dialogues that allow the WASD-specific (HTA) authentication databases to be administered. These include:

  creating databases
  deleting databases
  accessing databases for administering usernames
  listing usernames within databases
  adding usernames
  deleting usernames
  modifying username permissions and other data
  reseting in-server (cached) authentication information

  Chapter Chapter 3 covers authentication detail.
- **Site Log -** This accesses a plain-text file that could be used to record server or other significant site configuration changes if desired. Two methods of access are provided.

  1. Site-Log - open the file for editing, placing a date/time/author timestamp at the top
  2. Edit - open the file editing

  The file name and/or location may be specified using the logical name WASD_SITELOG.

Enabling Server Access

Many of the server activites listed above require server account write access to the directory in which the configuration files are stored. Where an autononmous scripting account is in use this poses minimal threat to server configuration integrity.

1. Specifically map the /wasd_root/local/ path and mark it as access always requiring authorization (ensure this is one on the first mappings in the file and certainly before any other /wasd_root/ ones).

```
# WASD_CONFIG_MAP
pass /wasd_root/local/* auth=all
```

2. Add appropriate authorization rules (example from "WASD Web Services - Install and Config").

```
# WASD_CONFIG_AUTH
["Web Admin"=WASD_WEBADMIN=id]
/httpd/-/admin/* r+w
/wasd_root/local/* r+w
```

3. Update access to the directory can be applied using the SECHAN utility (Section 13.12).

```
$ SECHAN /WRITE WASD_ROOT:[000000]LOCAL.DIR
$ SECHAN /WRITE WASD_ROOT:[LOCAL]
```

4. Load the new mapping and authorization rules.

```
$ HTTPD /DO=MAP
$ HTTPD /DO=AUTH=LOAD
```

Alternative Using /PROFILE

If a site is using SYSUAF authentication and security profiles enabled using the /PROFILE startup qualifier (Section 3.10.8) then a more restrictive set up is possible, retaining the default no-access to the [LOCAL] directory. This relies on the administering account(s) having read and write access to the [LOCAL] directory. It is then not necessary to grant that to the server account. It is possible to limit the application of VMS user profiles. This is an example.

```
# WASD_CONFIG_MAP
set /wasd_root/local/* profile auth=all
set * noprofile
```

To use this approach perform steps 1, 2 and 4 from above, substituting the following for step 3.

```
$ SECHAN /PACKAGE WASD_ROOT:[000000]LOCAL.DIR
$ SECHAN /PACKAGE WASD_ROOT:[LOCAL]
$ SECHAN /CONTROL WASD_ROOT:[000000]LOCAL.DIR
```

9.6 HTTPd Server Action

The server allows certain run-time actions to be initiated. Many of these functions can also be initiated from the command line, see Section 9.7.

When multiple servers are executing on a single node or within a cluster a JavaScript-driven checkbox appears in the bottom left of the administration menu. **Checking that box applies any subsequently selected action to all servers!**

Control Section

- **Server Restart/restartNOW/restartQuiet/Exit/exitNOW -** The difference between restart/exit and restartNOW/exitNOW is the former waits for any current requests to be completed, while the latter does it immediately regardless of any current connections. The restartQuiet variant continues processing until demand drops to zero for more than one second at which point it commences restart. If the browser has JavaScript enabled a cautionary alert requesting confirmation is generated (otherwise there is no confirmation).
- **Logging On/Off/Flush -** The WASD_CONFIG_LOG logical must be configured to allow access logging to be enabled and disabled from this menu.
- **Caching On/Off/Purge -** Caching may be enabled and disabled in an ad hoc fashion using these controls. When being disabled after being enabled all previous data is retained. If subsequently reenabled that data is then again available for use. This allows convenient assessment of the subject or even object benefits on the cahing. If purged all entries in the cache are removed.
- **Instance Startup -** An instance value may be set that overrides the configuration directive [InstanceMax] at next startup. This may be used to change the number of server processes on an ad hoc basis. Reset to "max" to return to configuration control. Note that this can be applied to the current node only or to all servers within a cluster, and that a subsequent restart is required.
- **/DO= Button and Field -** Provides a on-line facility parallel to that provided by the command-line /DO qualifier (Section 9.7). Any directive available via the command-line can be entered using this interface and applied on a per-node or per-cluster basis.

Configuration *Action* Section

- **Statistics Zeroed -** All counters are zeroed (except the *number-of-times-zeroed* counter!)
- **Mapping Rules Reload -** Reloads the path mapping rules from the on-disk file into the running server, clears the user SYSUAF mapping cache.

  Caution! If changing CGIplus script mapping it is advised to restart the server rather than reload. Some conflict is possible when using new rules while existing CGIplus scripts are executing.
- **Path Authorization Reload -** Reloads the path authorization directives from the on-disk file into the running server.

- **User Authentication Cache Purge -** For efficiency reasons authenticated user information is cached for a limited period within the running server. All this cached information may be completely purged using this action, forcing subsequent requests to be reauthenticated from the on-disk database.

9.7 HTTPd Command Line

A foreign command for the HTTPD control functionality will need to be assigned in the adminstration users' LOGIN.COM, for example:

```
$ HTTPD == "$WASD_EXE:HTTPD"
```

or (perhaps more likely)

```
$ HTTPD == "$WASD_EXE:HTTPD_SSL"
```

Some control of the executing server is available from the DCL command line on the system on which it is executing. This functionality, **via the /DO= qualifier**, is available to the privileged user. If a non-default server port then it will be necessary to provide a /PORT= qualifier with any command.

These directives are communicated from the command-line (and Server Administration page analogue - Control Section) to the per-node or per-cluster servers using the Distributed Lock Manager. On pre-VMS V8.2 the command buffer is limited to 15 bytes. From VMS V8.2 the buffer space available is 63 bytes. In a cluster all systems must support the larger buffer before WASD enables it. The smaller buffer space limits some of the directives that take free-form parameters (e.g. /DO=DCL=PURGE=USER=DANIEL).

Multi-Server/Cluster-Wide

If multiple servers are executing on a host or cluster it is possible to control all of them by adding the /CLUSTER or /ALL qualifiers. Of course, these commands are available from batch jobs as well as interactively. In a clustered WASD environment the same functionality is available via checkboxes from the online Server Administration facility.

9.7.1 Accounting

Server counters may be zeroed. These counters are those visible from the *statistics* Server Admininstration item and when using the HTTPDMON utility.

```
$ HTTPD /DO=ZERO
```

9.7.2 Alignment Faults

On Alpha and Itanium platforms alignment faults can be a significant performance issue and considerable effort has been invested in completely eliminating them. This was done using a internal reporting tool (primarily intended for the WASD developer) available from the Server Admin interface. Defining the logical name WASD_ALIGN_MAP to be a linker map of the build provides additional information.

```
$ HTTPD /DO=ALIGN=START
$ HTTPD /DO=ALIGN=STOP
$ HTTPD /DO=ALIGN=ZERO
$ HTTPD /DO=ALIGN=FAULT=1
```

9.7.3 Authentication

See Chapter 3.

The authorization rule file (HTTP$AUTH) may be reloaded using either of these variants.

```
$ HTTPD /DO=AUTH
$ HTTPD /DO=AUTH=LOAD
```

The authentication cache may be purged, resulting in re-authentication for all subsequent authorization-controlled accesses. This may be useful when disabling authorization or if a user has been locked-out due to too many invalid password attempts (Section 3.9).

```
$ HTTPD /DO=AUTH=PURGE
```

A "skeleton-key" username and password may be entered, amongst things allowing access to the Server Administration facility (Chapter 9).

```
$ HTTPD /DO=AUTH=SKELKEY=_<username>:<password>[:<period>]
```

9.7.4 Cache

Server cache control may also be exercised from the Server Administration page (Chapter 9). The file cache (see "WASD Web Services - Install and Config" document, "Cache Configuration" section) may be enabled, disabled and have the contents purged (declared invalid and reloaded) using

```
$ HTTPD /DO=CACHE=ON
$ HTTPD /DO=CACHE=OFF
$ HTTPD /DO=CACHE=PURGE
```

9.7.5 Configuration Check

Changes to configuration files can be validated at the command-line before reload or restart. This detects and reports any syntactical and fatal configuration errors but of course cannot check the *intent* of the rules.

```
$ HTTPD /DO=AUTH=CHECK
$ HTTPD /DO=CONFIG=CHECK
$ HTTPD /DO=GLOBAL=CHECK
$ HTTPD /DO=MAP=CHECK
$ HTTPD /DO=MSG=CHECK
$ HTTPD /DO=SERVICE=CHECK
```

The *config* check sequentially processes each of the *authorization*, *global*, *mapping*, *message* and *service* configuration files.

If additional server startup qualifiers are required to enable specific configuration features then these must also be provided when checking. For example:

```
$ HTTPD /DO=AUTH=CHECK /SYSUAF /PROFILE
```

9.7.6 DCL/Scripting Processes

These commands can be useful for flushing any currently executing CGIplus applications from the server, enabling a new version to be loaded with the next access. See "Scripting Environment" document.

All scripting processes, busy with a request or not, can be deleted (this may cause the client to lose data).

```
$ HTTPD /DO=DCL=DELETE
```

A gentler alternative is to delete idle processes and mark busy ones for deletion when completed processing.

```
$ HTTPD /DO=DCL=PURGE
```

For VMS V8.2 and later, a more selective DELETE and PURGE is possible. A user name, script name, or script file name can be supplied and only matching tasks have the specified action peformed.

```
$ HTTPD /DO=DCL=PURGE=USER=username
$ HTTPD /DO=DCL=PURGE=SCRIPT=script-path
$ HTTPD /DO=DCL=PURGE=FILE=script-file-name
```

9.7.7 DECnet Scripting Connections

All DECnet connections, busy with a request or not, can be disconnected (this may cause the client to lose data).

```
$ HTTPD /DO=DECNET=DISCONNECT
```

Purging is a better alternative, disconnecting idle tasks and marking busy ones for disconnection when complete.

```
$ HTTPD /DO=DECNET=PURGE
```

9.7.8 HTTP/2 Connection

Disconnect idle HTTP/2 connections.

```
$ HTTPD /DO=HTTP2=PURGE
```

All HTTP/2 connections can be disconnected (this may cause clients to lose data), or a specific connection number.

```
$ HTTPD /DO=HTTP2=PURGE=ALL
$ HTTPD /DO=HTTP2=PURGE=number
```

9.7.9 Instances

The number of server instances (see Section 8.1) may be set from the command line. This overrides any configuration file directive and applies at the next startup. Any configuration directive value may be used from the command line.

```
$ HTTPD /DO=INSTANCE=MAX
$ HTTPD /DO=INSTANCE=CPU
$ HTTPD /DO=INSTANCE=integer
```

Note that the server must be restarted for this to take effect, that this can be applied to the current node only or to all servers within a cluster, and that it remains in effect until explicitly changed to "MAX" allowing the WASD_CONFIG_GLOBAL configuration directive [InstanceMax] to once again determine the number of instances required. The same functionality is available from the Server Administration page (Section 9.6).

There are also directives to assist with WATCH activities (Section 10.1).

```
$ HTTPD /DO=INSTANCE=PASSIVE
$ HTTPD /DO=INSTANCE=ACTIVE
```

9.7.10 Logging

Server logging control may also be exercised from the server administration menu (Chapter 9).

Open the access log file(s).

```
$ HTTPD /DO=LOG=OPEN
```

Close the access log file(s).

```
$ HTTPD /DO=LOG=CLOSE
```

Close then reopen the access log file(s).

```
$ HTTPD /DO=LOG=REOPEN
```

Unwritten log records may be flushed to the file(s).

```
$ HTTPD /DO=LOG=FLUSH
```

9.7.11 Mapping

See "WASD Web Services - Install and Config" .

The mapping rule file (WASD_CONFIG_MAP) may be reloaded using either of these variants.

```
$ HTTPD /DO=MAP
$ HTTPD /DO=MAP=LOAD
```

9.7.12 Network Connection

Disconnect idle (persistent) connections.

```
$ HTTPD /DO=NET=PURGE
```

All network connections can be disconnected (this may cause clients to lose data), a specific connection number and those matching the specified URI.

```
$ HTTPD /DO=NET=PURGE=ALL
$ HTTPD /DO=NET=PURGE=number
$ HTTPD /DO=NET=PURGE=URI=pattern
```

Additionally, network connection acceptance can be suspended (leaving in-progress requests to complete), suspended and in-progress disconnected, and resumed.

```
$ HTTPD /DO=NET=SUSPEND
$ HTTPD /DO=NET=SUSPEND=NOW
$ HTTPD /DO=NET=RESUME
```

9.7.13 Shutdown and Restart

Server shutdown may also be exercised from the Server Administration page (Chapter 9).

The server may be shut down, without loss of existing client requests. Connection acceptance is stopped and any existing requests continue to be processed until conclusion.

```
$ HTTPD /DO=EXIT
```

The server may be immediately and unconditionally shut down.

```
$ HTTPD /DO=EXIT=NOW
```

The server may be restarted, without loss of existing client requests. Connection acceptance is stopped and any existing requests continue to be processed until conclusion. This effectively causes the server to exit normally and the DCL *wrapper* procedure to restart it.

```
$ HTTPD /DO=RESTART
```

The *now* variant restarts the server immediately regardless of existing connections.

```
$ HTTPD /DO=RESTART=NOW
```

The when-*quiet* variant restarts the server whenever request processing drops to zero for more than one second. It allows (perhaps non-urgent) changes to be put into effect through restart when everything has gone "quiet" and no demands are being placed on the server.

```
$ HTTPD /DO=RESTART=QUIET
```

9.7.14 Secure Sockets Layer

If the optional SSL component is installed and configured these directives become effective.

If X.509 authentication is enabled the Certificate Authority (CA) verification list can be reloaded.

```
$ HTTPD /DO=SSL=CA=LOAD
```

If a private key password is not included with the encode key it is requested by the server during startup. The following example shows the directive and the resulting prompt. When entered the password is not echoed.

```
$ HTTPD /DO=SSL=KEY=PASSWORD
Enter private key password []:
```

9.7.15 Throttle

Unconditionally release all queued requests for immediate processing.

```
$ HTTPD /DO=THROTTLE=RELEASE
```

Unconditionally terminate all requests queued waiting for processing. Clients receive a 503 "server too busy" response.

```
$ HTTPD /DO=THROTTLE=TERMINATE
```

For VMS V8.2 and later, a more selective RELEASE and TERMINATE is possible. A user name or script name can be supplied and only matching requests have the specified action peformed.

```
$ HTTPD /DO=THROTTLE=TERMINATE=REMOTE=pattern
$ HTTPD /DO=THROTTLE=TERMINATE=SCRIPT=pattern
```

9.7.16 WebSocket

Unconditionally disconnects all WebSocket applications.

```
$ HTTPD /DO=WEBSOCKET=DISCONNECT
```

For VMS V8.2 and later, more selective disconnects are possible. Disconnects WebSocket applications with connection number, with matching script names, and with matching scripting account usernames, respectively.

```
$ HTTPD /DO=WEBSOCKET=DISCONNECT=number
$ HTTPD /DO=WEBSOCKET=DISCONNECT=SCRIPT=pattern
$ HTTPD /DO=WEBSOCKET=DISCONNECT=USER=pattern
```

Chapter 10

WATCH Facility

The WATCH facility is a powerful adjunct in server administration. From the Server Administration facility (Chapter 9) it provides an **online, real-time, in-browser-window view of request processing in the running server**. The ability to observe live request processing on an ad hoc basis, without changing server configuration or shutting-down/restarting the server process, makes this facility a great configuration and problem resolution tool. It allows (amongst other uses)

assessment of mapping rules
assessment of authorization rules
investigation of request processing problems
observation of script interaction
general observation of server behaviour

A single client per server process can access the WATCH facility at any one time. It can be used in one of two modes.

- As a *one-shot*, one-off WATCH of a particular request. This is available from the *Request Report* page of the Server Administration facility. In this case the single indicated request is tagged to be WATCHed in all categories (see below) for the duration of the request (or until the client stops WATCHing).
- As described in the following chapter the server and all new requests being processed are candidates for being WATCHed. Categories are selected before initiating the WATCH and the report can be generated for a user-specified number of seconds or aborted at any time using the browser's *stop* button.

Options immediately below the duration selector allows the WATCH output to concurrently be included in the server process log. This allows a permanent record (at least as permanent as server logs) to be simply produced.

10.1 Server Instances

With a single instance (see Section 8.1) access to WATCH is always through the one server process. If multiple instances are configured WATCH requests, in common with all others, will be serviced by any one of the associated processes depending on the momentary state of the round-robin distribution.

This is often an issue for request WATCHing. The simplest scenario involves two instances. When the WATCH report is activated it will be serviced by the first process, when the request wishing to be WATCHed is accessed it (in the absence of any other server activity) will be serviced by the other process and will not be reported by WATCH on the first.

The solution is to suspend the round-robin request processing for the period of the WATCH activity. This does not shut any instance down but instead makes all but the supervisor instance quiescent. (Technically, it dequeues all the listening I/Os from non-supervisor instance server sockets, making the TCP/IP network driver send all connection requests to the one instance left with listening I/Os.) It is just a matter of making the non-supervisor instances active again when the WATCH activity is concluded.

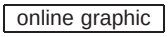
This may be done from the command-line using

```
$ HTTPD /DO=INSTANCE=PASSIVE
$ HTTPD /DO=INSTANCE=ACTIVE
```

or using the Server Administration facility (Chapter 9) where there are [Active] and [Passive] buttons available when multiple instances are in use. Neither transition disrupts any requests being established or in-progress.

10.2 Event Categories

An *event* is considered any significant point for which the server code has a reporting call provided. These have been selected to provide maximum information with minimum clutter and impact on server performance. Obvious examples are connection acceptance and closure, request path resolution, error report generation, network reads and writes, etc. Events are collected together into groupings to allow clearly defined areas of interest to be selected for reporting.



The report menu provides for the inclusion of any combination of the following categories.

Request

- **Processing -** Each major step in a request's progress. For example, path resolution and final response status.
- **Header -** Provides the HTTP request header as a section of blank-line terminated text.
- **Body -** The content (if a POST or PUT method) of the request. This is provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line.

Response

- **Processing -** Each major step in generating a response to the request. These generally reflect calls to a major server module such as file CACHE, FILE access, INDEX-OF, SSI processing, etc. One or more of these events may occur for each request. For instance a directory listing will show an INDEX-OF call and then usually a FILE call as any read-me file is accessed.
- **Header -** The blank-line terminated HTTP header to the response. Only server-generated headers are included. Scripts that provide a full HTTP stream do not have the header explicitly reported. The response body category must be enabled to observe these (indicated by a STREAM notation).
- **Body -** The content of the response. This is provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line. Some requests also generate very large responses which will clutter output. Generally this category would be used when investigating specific request response body problems.

General

- **Connection -** Each TCP/IP connection acceptance and closure. The connect shows which service the request is using (scheme, host name and port).
- **Path Mapping -** This, along with the authorization report, provides one of the most useful aspects of the WATCH facility. It comprises an event line indicating the path to be mapped (it can also show a VMS file specification if a *reverse-mapping* has been requested). Then as each rule is processed a summary showing current path, match "Y"/"N" for each path template and any conditional, then the result and conditional. Finally an event entry shows the resulting path, VMS file specification, any script name and specification resolved. The path mapping category allows the administrator to directly assess mapping rule processing with live or generated traffic.
- **Authorization -** When authorization is deployed this category shows the rules examined to determine if a path is controlled, any authentication events in assessing username and password, and the consequent group, user and request capabilities (read and/or write) for that path. No password information is displayed.
- **Error -** The essential elements of a request error report are displayed. This may include a VMS status value and associated system message.
- **CGI -** This category displays the generated CGI variable names and values as used by various forms of scripting and by SSI documents, as well as the processing of the response header returned by scripts.
- **DCL -** Debugging scripts can sometimes present particular difficulties. This category may help. It reports on all input/output streams with the process (SYS$INPUT, SYS$OUTPUT, SYS$COMMAND, CGIPLUSIN).
- **DECnet -** For the same reason as above this category reports all DECnet scripting input/output of the DECnet link. In particular, it allows the observation of the OSU scripting protocol.

- **WebDAV -** Provides WebDAV specific processing points including request and meta-data XML associated with resources.

Network

- **Activity -** For each raw network read and write the VMS status code and size of the I/O is recorded.
- **Data -** For each raw network read or write the contents are provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line.
- **HTTP/2 -** Provides a detailed overview of the underlying HTTP/2 framing and connection management exchanges between client and server. See HTTP/2 and WATCH for further detail.

Other

- **Logging -** Access logging events include log open, close and flush, as well as request entries.
- **Match -** Shows a significant level of detail during string matching activities. May be useful during mapping, authorization and conditional processing.
- **Script -** Sets CGI variable WATCH_SCRIPT allowing a script to explicitly detect this so as to output specific debugging or other information when being WATCHed.
- **SSL -** If the Secure Sockets Layer image is in use this category provides a indication of high-level activity.
- **Internal -** Includes information on other significant internal server processing. Examples are dictionary entries at various stages of request processing, and the high-level timing and timeout events occuring within that processing and the server in general.

Proxy

- **Processing -** Each major step during the serving of a proxied request.
- **Request Header -** The proxy server rebuilds the request originally received from the client. This category shows that rebuilt request, the one that is sent to the remote server.
- **Request Body -** In the case of HTTP POST or PUT methods any request body is displayed. This is provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line.
- **Response Header -** The blank-line terminated HTTP header to the response from the remote, proxied server.
- **Response Body -** The content of the response sent from the remote server. This is provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line.

- **Cache -** When proxy caching is enabled this category provides information on cache reading (serving a request from cache) and cache loading (writing a cache file using the response from a remote server). It will provide a reason for any request or response it does not cache, as well as report errors during file processing.
- **Cache Maintenance -** This category is not related to request processing. It allows routine and reactive cache purging activities to be watched.

Code Modules

If the server has been compiled using the WATCH_MOD=1 macro a set of module WATCHing statements is included. These provide far more detailed processing information than available with the generic WATCH, are intended primarily for debugging the server during development and testing. This is considered a specialized tool, with the quantity and level of detail produced most likely proving counter-productive in addressing general site configuration issues. The module items are shown below the usual WATCH items.

10.3 Request Filtering

By default all requests to all services are WATCHed. Fine control may be exercised over exactly which requests are reported, allowing only a selected portion of all requests being processed to be concentrated on, even on a live and busy server. This is done by *filtering* requests according the following criteria.

- **Protocol -** The HTTP protocol being used to transport the request. Multiple protocols may be selected and concurrently filtered against.
- **Client -** The originating host name or address. Unless server DNS host name resolution is enabled this must be expressed in dotted-decimal notation.
- **Service -** The service connected to. This includes the *scheme* of the service (i.e. “http:”, “https:”), the host name (real or virtual), and the port. The host name is the *official* name of the service as reported during server startup. As the port number is a essential part of the service specification it must always be explicitly supplied or wildcarded.
- **Request -** This filter operates on the entire HTTP request header. All fields supplied with the request are available to be filtered against. As this is a large, multi-line dataset filters can become quite complex and regular expression (see “WASD Web Services - Install and Config” document, “String Matching” section) matching may be useful (see examples below).
- **Path/Track -** Either, the request path, or a specific track identifier string. A path may be specified with a leading “/” for local paths or if WATCHing proxy requests with a full, or part of a full, URL. To WATCH requests associated with a particular access track (see “WASD Web Services - Install and Config” document, “Access Tracking” section) enter the track’s unique identifier string preceded by a dollar symbol (e.g. “$ORoKJAOef8sAAAkuACc”).
- **Realm & User -** This filters against request authentication information. As authorization occurs relatively late in request processing some data reported earlier by WATCH will not be available.

- **HTTP Status -** This allows a class of response status (1 (informational), 2 (success), 3 (redirection), 4 (client error) and 5 (server error)) or a specific response status (e.g. 200 (success), 404 (not found), 503 (service unavailable), etc.) to be filtered into the WATCH report. As this happens very late in request processing the number of reported events are limited but may provide some insight into particular processing problems.

In addition there are ***in* and *out* selectors** against each of the filters which include or exclude the particular request based on it matching the filter.

These filters are controlled using fully-specified, wildcarded strings or using regular expression patterns (see "WASD Web Services - Install and Config" document, "Request Processing Configuration" section). In common with all WASD processing, filter matching is case-insensitive. Of course, due to the point of application of a particular filter during request processing, some information may or may not be displayed. When a request is into or out of the report because of a matching filter a FILTER informational item is reported.

Examples

1. This first example shows various strings and patterns that could be applied to the client filter.

```
alpha.example.com
*.example.com
131.185.250.202
131.185.250.*
^10.68.250.*|10.68.251.*
```

2. This example various filters applied to the service (virtual server).

```
beta.example.com:8000
beta.example.com:*
http://*
https:*
*:80
```

3. The request filter contains the entire HTTP request header. This includes multiple, newline-delimited fields. Filtering can be simple or quite complex. These examples filter all POST requests (either in or out of the report depending on the respective selector), and all POSTs to the specified script respectively.

```
POST *
POST /cgi-bin/example*
```

   These are the equivalent regular expressions but also will stop comparing at the end of the initial request line. The second, in this case, will also only filter against HTTP/1.1 version requests (note the final period matching the <CR> of the <CR><LF> carriage control).

```
^^POST .*$
^^POST */cgi-bin/example *HTTP/1\.1.$
```

   This example uses a regular expression to constrain the match to a single header field (line, or newline-delimited string), matching all requests where the user agent reports using the "Gecko" browser component (Mozilla, Firefox, etc.)

```
^^User-agent:.*Gecko.*$
```

4. The path and track filter. The path contains a proxied origin server request and so can be used to filter proxy requests to specific sites.

```
/wasd_root/src/*
/cgi-bin/*
/web/*/cyrillic/*
$ORoKJAOef8sAAAkuACc
http://proxied.host.name/*
```

5. The authentication filters, realm and user, can be used to select requests for a particular authenticated user, all authenticated requests or all non-authenticated requests, amongst other application. The realm field allows the authenticated user to be further narrowed as necessary. All of the following examples show only the user field with the default *in* selector set.

   Authenticated requests for user DANIEL.

```
DANIEL
```

   All authenticated requests.

```
%*
```

10.4 Report Format

The following example illustrates the format of the WATCH report. It begins with multi-line heading. The first two record the date, time and official server name, with underline. The third provides the WASD server version. The fourth provides some TCP/IP agent information. Lines following can show OpenSSL version (if deployed), system information, server startup command-line, and then current server process quotas. The last three lines of the header provide a list of the categories being recorded, the filters in use, and the last, column headings described as follows:

time the event was recorded
the **module** name of the originating source code
the **line** in the code module
a unique **item** number for each thread being WATCHed
event **category** name
free-form, but generally interpretable **event** data

online graphic

Note that some items also include a block of data. The request header category does this, providing the blank-line terminated text comprising the HTTP header. Rule mapping also provides a block of information representing each rule as it is interpreted. Generally WATCH-generated information can be distinguished from other data by the uniform format and delimiting vertical bars. Initiative and imagination is sometimes required to interpret the free-form data but a basic understanding of HTTP serving and a little consideration is generally all that is required to deduce the essentials of any report. (Report manually wrapped for completeness.)

```
HTTPd-WASD/11.0.0 OpenVMS/AXP SSL (24-FEB-2016 12:02:29.57)
HP TCPIP$IPC_SHR V5.7-ECO1 (21-MAY-2010 14:44:46.64)
OpenSSL 1.0.2f  28 Jan 2016 (30-JAN-2016 09:50:05.33) [WASD_ROOT.SRC.OPENSSL-\
1_0_2F.INCLUDE.OPENSSL]*.H WASD [ALPHA.EXE.SSL]SSL_LIBSSL32.OLB
$ CC (V8.3/70390009) /DECC /STAND=RELAXED_ANSI /PREFIX=ALL /OPTIMIZE /NODEBUG \
/WARNING=(NOINFORM,DISABLE=(PREOPTW)) /FLOAT=IEEE /IEEE=DENORM /DEFINE=(WASD_VMS_V7,SESOLA,WA
Digital Personal WorkStation  with 1 CPU and 1536MB running VMS V8.3 (ODS-5 \
enabled, VMS NAML, VMS FIB, ZLIB disabled, REGEX enabled, lksb$b_valblk[64])
$ HTTPD /PRIORITY=4 /SYSUAF=(ID,SSL)/PERSONA=RELAXED/PROFILE
AST:1983/2000 BIO:1986/2000 BYT:49966656/49999424 DIO:5000/5000 ENQ:453/500 \
FIL:294/300 PGFL:451216/500000 PRC:0/100 TQ:97/100
DCL Scripting: detached, as HTTP$NOBODY, PERSONA enabled
Process: WASD:80 OTHER $1$DKA0:[wasd_root.][STARTUP]STARTUP_SERVER.COM;44
$1$DKA0:[wasd_root.][LOG_SERVER]KLAATU_20160224171154.LOG;1
Instances: KLAATU::WASD:80
Watching: connect, request, req-header, response, res-header, error (603) via HTTP/2
Filter: NONE
|Time_______|Module__|Line|Item|Category__|Event...|
|01:57:59.30 WATCH    1713 0001 CONNECT    HTTP/2 with 192.168.1.2,58855 on \
https://wasd.*****.***,443 (0.0.0.0)|
|++++++++++++++++++++++++++++++++++++++++
|01:57:59.30 HTTP2REQ 0235 0001 REQ-HEADER HEADER 463 bytes|
GET /httpd/-/admin/ HTTP/1.1
cache-control: max-age=0
authorization: Basic ************************
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 \
(KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36
dnt: 1
accept-encoding: gzip, deflate, sdch
accept-language: en-US,en;q=0.8
host: wasd.*****.***:443

|01:57:59.32 REQUEST  2980 0001 REQ-HEADER DATA|
ENTRY 001 [012] $ {12}request_line={28}GET /httpd/-/admin/ HTTP/1.1
ENTRY 002 [024] > {13}cache-control={9}max-age=0
ENTRY 003 [031] > {13}authorization={30}Basic ************************
ENTRY 004 [014] > {6}accept={74}text/html,application/xhtml+xml,\
application/xml;q=0.9,image/webp,*/*;q=0.8
ENTRY 005 [010] > {25}upgrade-insecure-requests={1}1
ENTRY 006 [001] > {10}user-agent={121}Mozilla/5.0 (Macintosh; Intel Mac OS X \
10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36
ENTRY 007 [004] > {3}dnt={1}1
ENTRY 008 [018] > {15}accept-encoding={19}gzip, deflate, sdch
ENTRY 009 [007] > {15}accept-language={14}en-US,en;q=0.8
ENTRY 010 .024. > {4}host={22}wasd.*****.***:443
|01:57:59.35 SERVICE  1721 0001 CONNECT    VIRTUAL wasd.*****.***:443|
|01:57:59.35 REQUEST  3640 0001 REQUEST    GET /httpd/-/admin/|
|01:57:59.38 ADMIN    0250 0001 RESPONSE   ADMIN /httpd/-/admin/|
|01:57:59.39 NET      2308 0001 RES-HEADER DATA|
ENTRY 001 .018. $ {15}response_status={3}200
ENTRY 002 [028] < {12}x-user-agent={121}Mozilla/5.0 (Macintosh; Intel Mac OS X \
10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36
ENTRY 003 .011. < {6}server={34}HTTPd-WASD/11.0.0 OpenVMS/AXP SSL
ENTRY 004 [002] < {4}date={29}Wed, 24 Feb 2016 15:27:59 GMT
ENTRY 005 .005. < {13}accept-ranges={5}bytes
ENTRY 006 [008] < {15}accept-encoding={13}gzip, deflate
ENTRY 007 [020] < {25}strict-transport-security={16}max-age=16416000
```

```
ENTRY 008 .004. < {7}expires={29}Fri, 13 Jan 1978 14:00:00 GMT
ENTRY 009 [030] < {13}cache-control={18}no-cache, no-store
ENTRY 010 .028. < {6}pragma={8}no-cache
ENTRY 011 .030. < {12}content-type={29}text/html; charset=ISO-8859-1
ENTRY 012 [006] < {14}content-length={5}15719
|01:57:59.40 HTTP2REQ 0713 0001 RES-HEADER HEADER 497 bytes|
HTTP/1.1 200 OK
x-user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.\
36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36
server: HTTPd-WASD/11.0.0 OpenVMS/AXP SSL
date: Wed, 24 Feb 2016 15:27:59 GMT
accept-ranges: bytes
accept-encoding: gzip, deflate
strict-transport-security: max-age=16416000
expires: Fri, 13 Jan 1978 14:00:00 GMT
cache-control: no-cache, no-store
pragma: no-cache
content-type: text/html; charset=ISO-8859-1
content-length: 15719

|01:57:59.41 REQUEST  1143 0001 REQUEST    STATUS 200 (OK) rx:29 tx:15809 \
bytes 0.109368 seconds 144,813 B/s|
|----------------------------------------
|01:58:50.58 end|
```

10.5 Usage Suggestions

The following provides a brief explanation on the way WATCH operates and any usage implications.

A single client may be connected to the WATCH facility at any given time. When connecting the client is sent an HTTP response header and the WATCH report heading lines. The request then remains connected until the WATCH duration expires or the client overtly aborts the connection. During this period the browser behaves as if receiving a sometimes very slow, sometimes stalled, plain-text document. As the server processes WATCHable events the text generated is sent to the WATCH-connected client.

If the connection is aborted by the user some browsers will consider document retrieval to be incomplete and attempt to reconnect to the service if an attempt is made to print or save the resulting document. As the printing of WATCH information is often quite valuable during problem resolution this behaviour can result in loss of information and generally be quite annoying. Appropriate use of the duration selector when requesting a report can work around this, as at expiry the *server* disconnects, browsers generally interpreting this as legitimate end-of-document (when no content-length has been specified).

During report processing some browsers may not immediately update the on-screen information to reflect received data without some application activity. If scroll-bars are present on the document window manipulating either the horizonal or vertical slider will often accomplish this. Failing that minimizing then restoring the application will usually result in the most recent information being visible.

Browser *reload*/*refresh* may be used to restart the report. A browser will quite commonly attempt to remain at the current position in the document, which with a WATCH report's sustained but largely indeterminate data stream may take some time to reach. It is suggested

the user ensure that any vertical scroll-bar is at the beginning of the current report, then refresh the report.

Selecting a large number of categories, those that generate copious output for a single event (e.g. response body) or collecting for extended periods can all result in the receipt of massive reports. Some browsers do not cope well with documents megabytes in size.

Note

WATCH reports are written using blocking I/O (with HTTP/1.*n*, HTTP/2 is completely asynchronous). This means when large bursts of data are being generated (e.g. when WATCHing network data, response bodies, etc.) significant granularity may be introduced to server processing. Also if the WATCH client fails or blocks completely server processing could halt completely! (This has been seen when WATCHing through a firewall.)

When supplying WATCH output as part of a problem report please ZIP the file and include it an an e-mail attachment. Mailers often mangle the report format making it difficult to interpret.

10.6 Command-Line Use

Although intended primarily as a tool for online use WATCH can be deployed at server startup with a command-line qualifier and provide report output to the server process log. This is slightly more cumbersome than the Web interface but may still be useful in some circumstances. Full control over event categories and filters is possible.

- **/NOWATCH** Disables the use of the online WATCH facility.
- **/WATCH=** Enables the server WATCH facility, dumping to standard output (and the server process log if detached). When in effect the online facility is unavailable. The string supplied to the qualifier may comprise four comma-separated components. Only the first is manadatory. Stated order is essential. It will probably be necessary to enclose the complete string in quotation marks.
  - **LIST -** The LIST keyword provides a list of all the categories (items) available for WATCHing.
  - **NOSTARTUP -** This keyword suppresses WATCH output until the server is ready to process requests. It must be the leading keyword.
  - ***items* -** A parenthesized, comma-separated list of category keywords. Available keywords can be displayed using the LIST facility.
  - ***filters* -** A client, service and path filters can be provided following the specification of required items. They must be provided in the order listed above. Leading filters that are not required must be provided as single, asterisk wildcards. WATCH parameter with filters containing forward-slashes will require quoting.

The following examples illustrate the command-line WATCH specification.

```
/NOWATCH
/WATCH=NOSTARTUP,ITEMS=(REQUEST,RESPONSE,MAPPING)
/WATCH="ITEMS=(REQUEST,RESPONSE,ERROR),*,*,/cgi-bin/*"
/WATCH=LIST
```

Chapter 11
Server Performance

The server has a single-process, multi-threaded, asynchronous I/O design. On a single-processor system this is the most efficient approach. On a multi-processor system it is limited by the single process context (with scripts executing within their own context). For I/O constrained processing (the most common in general Web environments) the AST-driven approach is quite efficient.

WASD v10 Data

Please note that relative performance data has not yet been generated for WASD v11. What follows is v10 data. WASD v11 is very similar in most respects, especially for HTTP/1.1 processing, and so the relativities here likely remain valid even if the absolute numbers might be marginally different.

As also related in Section 5.2, on the developer's bench using the same (HTTP/1.1) load profile** with WASD v11.0 compared to v10.4 showed ~5% additional CPU and duration (with v11.0). This is (probably) largely due to dictionary processing.

*** 100 individual files, size 2kB to 250kB, 50 concurrent, ~30% CPU utilisation (~5% USER mode, mostly INTERRUPT servicing), batched 10,000 at a time over a LAN.*

The test-bench system was an **HP rx2600 (Itanium 1.40GHz/1.5MB) with 2 CPUs and 8191MB**, running VMS V8.3-1H1 and HP TCP/IP Services Version V5.6 - ECO 2.

Many thanks to Kednos (http://www.kednos.com) for the use of the system.

The performance data is collected using the "WASDbench" utility (Section 13.14). Previous performance measurements had been made using the ApacheBench utility (Section 13.6) but experimenting with both it was observed that (perhaps the asynchronous I/O of) WASDbench provided generally greater throughput and less variation on this higher performance Itanium platform. DCL procedures with sets of WASDbench calls are used to benchmark requests. These procedures and the generated output from benchmark runs (collected via $@*procedure*/OUTPUT=*filename*) are available in the WASD_ROOT:[EXERCISE] directory.

These results are indicative only!

On a multi-user system too many things vary slightly all the time.

Every endeavour has been made to ensure the comparison is as equitable as possible. Each server executes at the same process priority, access logging and host name lookup disabled, and runs on the same machine in the same relatively quiescent environment. Each test run was interleaved between each server to try and distribute any environment variations. Those runs that are very high throughput use a larger number of requests to improve sample period validity. All servers were configured pretty-much "out-of-the-box", minimal changes (generally just enough to get the test environment going). The server and test-bench utility were located on the same system eliminating actual data on the wire. Multiple data collections have yielded essentially the same relative results.

For the test-bench WASD v10.0 is present on port 7080.

OSU Comparison

The OSU comparison used the v3-11 release suitable for Itanium. OSU is executing in kernel-threads mode ("M"). OSU is present on port 7777.

Apache Comparison

The Apache comparison used the latest CSWS (CSWS-V0201, based on v2.0.52), Perl (CSWS_PERL-V0201) and PHP (CSWS_PHP-V0210) kits, and any required updates/ECOs, available at the time of collection. Apache is present on port 8888.

11.1 Simple File Request Turn-Around

A series of tests using batches of accesses. The first test returned an empty file measuring response and file access time, without any actual transfer. The second requested a file of 64K characters, testing performance with a more realistic load. All were done using one and ten concurrent requests. Note that the Apache measurement is "out-of-the-box" - the author could find no hint of a file cache, let-alone how to enable/disable one. Each request required a complete TCP connection and disposal.

Cache Disabled

Concurrency 1 - Requests/Second

Response	WASD	OSU	Apache
0K	137	90	32
64K	126	63	33

Concurrency 10 - Requests/Second

Response	WASD	OSU	Apache
0K	112	96	28
64K	98	75	29

Cache Enabled

Concurrency 1 - Requests/Second

Response	WASD	OSU	Apache
0K	1606	1252	27
64K	862	562	30

Concurrency 10 - Requests/Second

Response	WASD	OSU	Apache
0K	1730	1580	29
64K	982	672	28

Result files:

WASD_ROOT:[EXERCISE]PERF_FILES_NOCACHE_WB_V10.TXT
WASD_ROOT:[EXERCISE]PERF_FILES_WB_V10.TXT

The difference between cached and non-cached result with the zero file size (no actual data transfer involved) gives some indication of the raw difference in response latency, some 5x improvement. It also indicates the relative efficiencies of file-system access. This is a fairly BASIC analysis, but does give an approciation of the utility and efficiencies of having an in-server cache.

File Transfer Rate

Requests for a large *binary* file indicate a **potential transfer rate of many tens of Mbytes per second**. On the test-bench this data does not get onto the wire of course but it does serve to demonstrate that server architecture should not be the limiting factor in file throughput.

Transfer Rate - MBytes/Second

Response	Concurrent	WASD	OSU	Apache
13MB (26134 blocks)	1	113	71	85
13MB (26134 blocks)	10	115	58	93

Result file:

WASD_ROOT:[EXERCISE]PERF_XFER_WB_V10.TXT

File Record Format

The WASD server can handle STREAM, STREAM_LF, STREAM_CR, FIXED and UNDEFINED record formats very much more efficiently than VARIABLE or VFC files. With STREAM, FIXED and UNDEFINED files the assumption is that HTTP carriage-control is within the file itself (i.e. at least the newline (LF), all that is required required by browsers), and does not require additional processing. With VARIABLE record files the carriage-control is implied and therefore each record requires additional processing by the server to supply it. Even with variable record files having multiple records buffered by the HTTPd before writing them collectively to the network improving efficiency, stream and binary file reads are by Virtual Block and are written to the network immediately making the transfer of these very efficient indeed!

CPU Consumed

Just one other indicative metric; CPU time consumed during the file request runs. The value for Apache was not measured as it would be distributed over an indeterminate number of child processes.

CPU Time Consumed (Seconds)

Cache	WASD	OSU
Disabled	4.36	12.75
Enabled	1.51	3.32

Result files (towards end of each):

WASD_ROOT:[EXERCISE]PERF_FILES_NOCACHE_WB_V10.TXT
WASD_ROOT:[EXERCISE]PERF_FILES_WB_V10.TXT

11.2 Scripting

A simple performance evaluation shows the relative merits of the four WASD scripting environments available, plus a comparison with OSU and Apache.
WASD_ROOT:[SRC.CGIPLUS]CGIPLUSTEST.C, which executes in both standard CGI and CGIplus environments, and an ISAPI example DLL, WASD_ROOT:[SRC.CGIPLUS]ISAPIEXAMPLE.C, which provides equivalent output. A series of accesses were made. The first test returned only the HTTP header, evaluating raw request turn-around time. The second test requested a body of 64K characters, again testing performance with a more realistic load.

The CGIPLUSTEST.C with the v10 package has be reworked (primarily by using CGILIB) to provide a more equitable comparison by using CGI with WASD and Apache, and the native dialog phase (i.e. non-CGI) with OSU.

DECnet-based WASD scripting was tested using essentially the same environment as detached process based CGI, assessing the performance of the same script being executed using DECnet to manage the processes. The OSU-emulation provided by WASD was also (somewhat obviously) provided using DECnet.

Concurrency 1 - Requests/Second

Response	CGI	CGIplus	ISAPI	DECnet	WASD-OSU	OSU	Apache	Apache-OSU
0KB	47	714	557	32	46	48	3.9	4.0
64KB	58	418	263	31	39	38	3.0	1.7

Concurrency 10 - Requests/Second

Response	CGI	CGIplus	ISAPI	DECnet	WASD-OSU	OSU	Apache	Apache-OSU
0KB	62	979	785	47	60	55	5.5	4.2
64KB	60	377	228	42	61	53	4.4	4.7

Result file:

WASD_ROOT:[EXERCISE]PERF_SCRIPTS_WB_V10.TXT

Although these results are indicative only, they do show the persistent environment of CGIplus and ISAPI to have a potential for improvement over standard CGI with factors of 5x to 10x - a not inconsiderable improvement. Of course this test generates the output stream very simply and efficiently and so excludes any actual processing time that may be required by a “real” application. If the script or application has a large activation time the reduction in response latency could be even more significant (e.g. with scripting engines such as Perl, PHP and Python, and RDMS access languages); see Persistent Scripting Observations immediately below.

Persistent Scripting Observations

CGI scripting is notoriously slow (as illustrated above), hence the effort expended by designers in creating persistent scripting environments - those where the scripting engine (and perhaps other state) is maintained between requests. Both WASD and Apache implement these as integrated modules, the former as CGIplus/RTE, and in the latter as loadable modules.

The following comparison uses two of the most common scripting environments and engines shared between WASD and Apache, Perl and PHP. The engines used in both server environments were identical. No comparison is made with OSU (in part due to the lack of obvious integration of such environments with OSU).

A simple script for each engine is used as a common test-bench for the two servers.

```
<!-- face2face.php -->
<?php
echo "<B>Hello!</B>"
?>

# face2face.pl
print "Content-Type:  text/html\n\n
<B>Hello!</B>
";
```

These are designed to measure the script environment and its activation latencies, rather than the time required to process script content (which should be consistent considering they are the same engines). In addition, the standard *php_info.php* is used to demonstrate with a script that actually performs some processing.

Persistent Scripting - Requests/Second

	Concurrent	WASD	Apache
face2face.pl	1	169	6.9
face2face.pl	10	229	9.5
face2face.php	1	82	16
face2face.php	10	203	21
php_info.php	1	33	18
php_info.php	10	135	18

Result file:

WASD_ROOT:[EXERCISE]PERF_PERSIST_WB_V10.TXT

These results demonstrate the efficiency and scalability of the WASD CGIplus/RTE technology used to implement its persistent scripting environments. Most site-specific scripts can also be built using the libraries, code fragments, and example scripts provided with the WASD package, and obtain similar efficiencies and low latencies. See “WASD Web Services - Scripting” document.

11.3 SSL

At this time there are no definitive measurements of SSL performance (Chapter 4). One might expect that because of the CPU-intensive cryptography employed in SSL requests that performance, particularly where concurrent requests are in progress, would be significantly lower. In practice SSL seems to provide more-than-acceptable responsiveness.

11.4 Suggestions

Here are some suggestions for improving the performance of the server, listed in approximate order of significance. Many are defaults. Note that these will have proportionally less impact on an otherwise heavily loaded system.

1. Disable host name resolution (configuration parameter [DNSLookup]). **DNS latency can slow request processing significantly!** Most log analysis tools can convert literal addresses so DNS resolution is often an unnecessary burden.
2. Later versions of TCP/IP Services for OpenVMS seem to have large default values for socket send and receive buffers. MultiNet and TCPware are reported to improve transfer of large responses by increasing low default values for send buffer size. The WASD global configuration directives [SocketSizeRcvBuf] and [SocketSizeSndBuf] allow default values to be adjusted. WATCH can be used to report network connection buffer values.
3. Enable file caching (configuration parameter [Cache]).
4. Ensure served files are not VARIABLE record format (see above). Enable STREAM-LF conversion using a value such as 250 (configuration parameter [StreamLF], and SET against required paths using mapping rules).
5. Use persistant DCL/scripting processes (configuration parameter [ZombieLifeTime])
6. Ensure script processes are given every possible chance to persist (configuration parameter [DclBitBucketTimeout]).
7. Use the persistent scripting capabilities of CGIplus or ISAPI whenever possible.
8. Ensure the server account's WSQUO and WSEXTENT quotas are adequate. A constantly paging server is a slow server!
9. If the server is intended to provide significant numbers of larger files (e.g. multimedia) then setting [BufferSizeNetFile] can improve data rates. Experiments should determine the maximum value (30000 - 60000) that provides the best server data rate.
10. Tune the network and DCL output buffer size to the Maximum Transfer Unit (MTU) of the server's network interface. Using Digital TCP/IP Services (a.k.a. UCX) display the MTU.

```
TCPIP> SHOW INTERFACE
                                                          Packets
Interface   IP_Addr         Network mask          Receive          Send     MTU

 SE0        203.127.158.3   255.255.255.0          376960        704345    1500
 LO0        127.0.0.1       255.0.0.0                 306           306       0
```

   In this example the MTU of the ethernet interface is 1500 (bytes). Set the [BufferSizeNetWrite] configuration directive to be some multiple of this. In the case of 1500, say 3000, 4500 or 6000. Also set the [BufferSizeDclOutput] to the same value. Rationale: always use completely filled network packets when transmitting data.

   The [BufferSizeNetMTU] directive when set to the MTU will automatically optimise buffer sizes using this approach.

11. Disable logging (configuration parameter [Logging]).

12. Set the HTTP server process priority higher, say to 6 (use startup qualifier /PRIORITY=). Do this after due consideration. It will only improve response time if the system is also used for other, lower priority purposes. It will not help if Web-serving is the sole acitivity of the system.
13. Use a pre-defined log format (e.g. “common”, configuration parameter [LogFormat]). User-specified formats require more processing for each entry.
14. Disable request history (configuration parameter [RequestHistory]).
15. Disable activity statistics (configuration parameter [ActivityDays]).

Chapter 12

HTTPd Web Update

The **Upd**ate facility allows Web documents and file environments to be administered from a standard browser. This capability is available to Web administrator and user alike. Availability and capability depends on the authorization environment within the server.

It **should be stressed** that this is not designed as a full hypertext administration or authoring tool, and for document preparation relies on the editing capabilities of the <TEXTAREA> widget of the user's browser. It does however, allow **ad-hoc changes** to be made to documents fairly easily, as well as allowing documents to be deleted, and directories to be created and deleted.

Consult the current **Upd**ate documentation for usage detail.

online hypertext link

online graphic

online graphic

Update Access Permission

If SSL is in use (see "WASD Web Services - Install and Config") then username/password privacy of the authorization environment is inherently secured via the encrypted communications. To restrict web update functionality to this secure environment add the following to the WASD_CONFIG_MAP configuration file:

```
/upd/*  "403 Access denied."  ![sc:https]
```

Of course, the user must have write (POST/PUT) access to the document or area on the server (i.e. the *path*) and the server account have file system permission to write into the parent directory.

The server will report "Insufficient privilege or object protection violation ... /path/document" if it does not have file system permission to write into a directory.

Also see Section 3.13 for information on write access control for the server account.

Chapter 13
Utilities and Facilities

Foreign commands for external utilities (and the HTTPD control functionality) will need to be assigned from the adminstration users' LOGIN.COM either explicitly or by calling the WASD_ROOT:[EXAMPLE]WASDVERBS.COM procedure.

```
$ AB == "$WASD_EXE:AB"
$ HTTPD == "$WASD_EXE:HTTPD"
$ HTTPDMON == "$WASD_EXE:HTTPDMON"
$ MD5DIGEST == "$WASD_EXE:MD5DIGEST"
$ QDLOGSTATS == "$WASD_EXE:QDLOGSTATS"
$ SECHAN == "$WASD_EXE:SECHAN"
$ STREAMLF == "@WASD_EXE:STREAMLF"
$ WB == "$WASD_EXE:WB"
```

13.1 Echo Facility

Ever had to go to extraordinary lengths to find out exactly what your browser is sending to the server? The server provides a request echo facility. This merely returns the complete request as a plain-text document. This can be used for for checking the request header lines being provided by the browser, and can be valuable in the diagnosis of POSTed forms, etc.

This facility must be enabled through a mapping rule entry.

```
script /echo/* /echo/*
```

It may then be used with any request merely by inserting "/echo" at the start of the path, as in the following example.

```
http://www.example.com/echo/wasd_root/
```

13.2 Hiss Facility

The *hiss* facility provides a response stream made up of random alpha-numeric characters (a sort of alpha-numeric white-noise). No response header is generated and the stream will continue (by default) up to one megabyte of output, or until the client closes the connection. This maximum may be controlled my appending an integer representing the number of kilobytes maximum to the mapping. This facility must be enabled through a mapping rule entry and may then be used for specific requests.

```
map /**.dll* /hiss/64/*.dll*
map /**/system32/* /hiss/64/*/system32/*
map /**default.ida* /hiss/64/*default.ida*
script /hiss/* /hiss/*
```

Usage details are described in “WASD Web Services - Install and Config” .

13.3 Stream Facility

The *stream* facility provides a quantified or unlimited response stream of printable or binary octets. It is intended as a light-weight data source delivering content at the maximum throughput capable by the server and platform. This can be used as a test source or for end-to-end metrics. This facility must be enabled through a mapping rule.

```
script /stream/* /stream/*
```

It may then be used to generate streams of data with various characteristics and sizes by including parameters in the URL.

- Without parameters it produces a text/plain response header with unlimited stream of random 8 bit printable and newline characters. The stream ceases at client disconnection.

  ```
  http://www.example.com/stream/
  ```

- With an integer parameter the stream ceases when the response has delivered that many kilobytes (1024) of characters.

  ```
  http://www.example.com/stream/50/
  ```

- A 100 kilobyte stream of repeated 80 column, newline terminated characters in the range “+” (0x2b) to “z” (ox7a). Intended to provide an entirely predictable sequence for testing purposes.

  ```
  http://www.example.com/stream/text:100/
  ```

- The following produces an application/binary response header with unlimited stream of random octets.

  ```
  http://www.example.com/stream/binary/
  ```

- One megabyte of random octets.

  ```
  http://www.example.com/stream/binary:1024/
  ```

- An unlimited stream of octets cycling from 0x00 to 0xff. Intended to provide an entirely predictable sequence for testing purposes.

  ```
  http://www.example.com/stream/octets/
  ```

13.4 Where Facility

Need to locate where VMS has the HTTPd files? This simple facility maps the supplied path then parses it to obtain a resulting VMS file specification. **This does not demonstrate whether the path actually exists!**

This facility must be enabled through a mapping rule entry.

```
script /where/* /where/*
```

It may then be used with any request merely by inserting "/where" at the start of the path, as in the following example.

```
http://www.example.com/where/wasd_root/
```

13.5 Xray Facility

The Xray facility returns a request's complete response, **both header and body**, as a plain text document. Being able to *see* the internals of the response header as well as the contents of the body rendered in plain text can often be valuable when developing scripts, etc.

This facility must be enabled through a mapping rule entry.

```
script /Xray/* /Xray/*
```

It may then be used with any request merely by inserting "/xray" at the start of the path, as in the following example.

```
http://www.example.com/xray/wasd_root/
```

13.6 ApacheBench

This server stress-test and benchmarking tool, as used in the Apache Distribution, is included with the WASD package (sourced from http://webperf.zeus.co.uk/ab.c), within license conditions.

```
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd.
Copyright (c) 1998 The Apache Group.
```

ApacheBench will only compile and run for Alpha, Itanium or VAX systems with VMS 7.*n* or greater available. Also see the WASD analogue, Section 13.14. ApacheBench is a simple but effective tool, allowing a single resource to be requested from a server a specified number of times and with a specified concurrency. This can be used to benchmark a server or servers, or be used to stress-test a server configuration's handling of variable loads of specific resquests (before exhausting process quotas, etc.) This utility has remained at the 1.3 release due to subsequent versions (e.g. 2.0) having Apache API dependencies.

A small addition to functionality has been made. The WASD ApacheBench displays a count of the HTTP response categories received (i.e. the number of 2*nn*s, 4*nn*s, etc.) This allows easier assessment of the relevance of results (i.e. measuring performance of some aspect only to find the results showed the performance of 404 message generation - and yes, an annoying experience of the author's prompted the changes!)

The following examples illustrate its use.

```
$ AB -H
$ AB -C 10 -N 100 http://the.server.name/wasd_root/exercise/0k.txt
$ AB -C 50 -N 500 -K http://the.server.name/wasd_root/exercise/64k.txt
$ AB -C 10 -N 100 http://the.server.name/cgi-bin/cgi_symbols
```

13.7 CALogs

The Consolidate Access LOGS utility (pronounced similar to the breakfast cereal brand :-) merges multiple HTTP server common and combined format access logs into a single log file with records in time-order. Due to the granularity of HTTP server entry timestamps (one second) the records are sorted to the one second but not within the one second.

It uses RMS and the VMS sort-merge routines to provide the basic consolidation functionality. An RMS search uses the supplied wildcard log file specification. Matching files are opened and each record read. The date/time field is parsed and a binary timestamp generated. Records with formats or date/time fields that do not make sense to the utility are discarded. When all files have been processed the sort-merge is performed using the timestamp as the key. The sorted records are then written to the specified output file.

$ calogs <log-file-spec> [<output-file-name>] [<qualifiers>]

Parameters and Qualifiers

Parameter	Description
/HELP	basic usage information
/NOPROXY	discard proxy service records
/NOWASD	discard WASD server status/timestamp entries
/OUTPUT=	alternate method of specifying merged file name
/PROXY	discard non-proxy service records
/QUIET	no messages apart from errors
/VERBOSE	per-file progress messages
/VERSION	display the utility version and copyright message

Usage Examples

```
$ CALOGS == "$WASD_EXE:CALOGS"
$ CALOGS WASD_LOGS:*200205*.LOG 2002_MAY.LOG
$ CALOGS /VERBOSE WASD_LOGS:
$ CALOGS /NOWASD WASD_LOGS:*200206*.LOG_* /OUTPUT=2002_JUNE.LOG
$ CALOGS /PROXY /NOWASD WASD_LOGS:*2002*.LOG 2002_PROXY.LOG
```

13.8 HTAdmin

The HTAdmin utility assists in with the command-line maintenance of $HTA authorization databases. See “WASD Web Services - Install and Config” document, “Authorization Configuration” section, and Chapter 3.

$ htadmin <database> [<username>] [<qualifiers>]

Parameters and Qualifiers

Parameter	Description
/ADD	add a new record
/CONFIRM	confirm deletion of database
/CONTACT="<string>"	contact information for record
/CREATE	create a new database
/CSV[=TAB | char]	comma-separated listing (optional character)
/DATABASE=	database name (or as command-line parameter)
/DELETE	delete a database or username record from a database
/DISABLED	username record is disabled (cannot be used)
/EMAIL="<string>"	email address for record
/ENABLED	username record is enabled (can be used)
/FULL	listing showing full details
/GENERATE	generate a six character password
/HELP	basic usage information
/[NO]HTTPS	synonym for /SSL
/LIST	listing (brief by default, see /FULL and /CSV)
/MODIFY	synonym for /UPDATE
/NAME="<string>"	full name for username record
/OUTPUT=	alternate output for database listing
/PASSWORD[=<string>]	username record password (prompts if not supplied)
/PIN	generate four-digit "PIN number" for password
/[NO]READ	username can/can’t read
/SORT[=<parameters>]	sort the records into a new/another database
/[NO]SSL	user can only authenticate via SSL (“https:”)
/[NO]WRITE	username can/can’t write

Parameter	Description
/UPDATE	update an existing username record
/USER=<string>	username
/VERSION	display version of HTADMIN

Usage Examples

- To create a new database named EXAMPLE.$HTA (in the current directory)

```
$ HTADMIN EXAMPLE /CREATE
```

- Delete an existing database

```
$ HTADMIN EXAMPLE /DELETE /CONFIRM
```

- List (briefly) the records

$ HTADMIN EXAMPLE

- List (briefly) the specific user record DANIEL

```
$ HTADMIN EXAMPLE DANIEL
```

- List all detail (132 colums) of the specified user record

```
$ HTADMIN EXAMPLE DANIEL /FULL
```

- To add the new record DANIEL with default read access

```
$ HTADMIN EXAMPLE DANIEL /ADD /NAME="Mark Daniel"
```

- Add the new record DANIEL with contact details and read+write access

```
$ HTADMIN EXAMPLE DANIEL /ADD /WRITE /CONTACT="Postal Address"
```

- Add the new record DANIEL and be prompted for a password, or to specify the password on the command-line, or have the utility generate a password or four-digit PIN style password (which is displayed after the record is sucessfully added)

```
$ HTADMIN EXAMPLE DANIEL /ADD /NAME="Mark Daniel" /PASSWORD
$ HTADMIN EXAMPLE DANIEL /ADD /NAME="Mark Daniel" /PASSWORD=cher10s
$ HTADMIN EXAMPLE DANIEL /ADD /NAME="Mark Daniel" /GENERATE
$ HTADMIN EXAMPLE DANIEL /ADD /NAME="Mark Daniel" /PIN
```

- To update an existing record

```
$ HTADMIN EXAMPLE DANIEL /UPDATE /EMAIL="Mark.Daniel@wasd.vsm.com.au"
```

- Update the specified record's password (interactively) then to generate a four digit PIN for a password (which is then displayed)

```
$ HTADMIN EXAMPLE DANIEL /UPDATE /PASSWORD
$ HTADMIN EXAMPLE DANIEL /UPDATE /GENERATE
$ HTADMIN EXAMPLE DANIEL /UPDATE /PIN
```

- Disable then enable an existing user record without changing anything else

```
$ HTADMIN EXAMPLE DANIEL /UPDATE /DISABLE
$ HTADMIN EXAMPLE DANIEL /UPDATE /ENABLE
```

- To list the entire database, first briefly, then in 132 column mode (with all detail), then finally as a comma-separated listing

```
$ HTADMIN EXAMPLE
$ HTADMIN EXAMPLE /FULL
$ HTADMIN EXAMPLE /CSV
```

Sort Details

The /SORT qualifier sorts the current database records according to the /SORT= parameters. It can be used with the /LIST qualifier to produce ordered reports or will output the records into another authentication file. By default it sorts ascending by username. Qualifier parameters allow a sort by DATE or COUNT. Each of these allows the further specification of which date or count; ACCESS, CHANGE or FAILURE.

- Generating a listing with specified order

```
$ HTADMIN EXAMPLE /LIST /SORT=DATE=ACCESS
$ HTADMIN EXAMPLE /LIST /SORT=COUNT=FAILURE /OUTPUT=EXAMPLE.LIS
```

- Sort descending by username into a higher version of EXAMPLE.$HTA

```
$ HTADMIN EXAMPLE /SORT
```

- To sort by username into another .$HTA file

```
$ HTADMIN EXAMPLE /SORT /OUTPUT=ANOTHER
```

- List by most-recently accessed

```
$ HTADMIN EXAMPLE /LIST /SORT=DATE
```

- List by most-recently failed to authenticate

```
$ HTADMIN EXAMPLE /LIST /SORT=DATE=FAILURE
```

- Sort file into order by most frequently authenticated (accessed)

```
$ HTADMIN EXAMPLE /SORT=COUNT
```

13.9 HTTPd Monitor

The HTTP server may be monitored in real-time using the HTTPDMON utility.

online graphic

This utility continuously displays a screen of information comprising three or four of the following sections:

1. **Process Information**
   HTTPd process information includes its up-time, CPU-time consumed (excluding any subprocesses), I/O counts, and memory utilization. The “Servers:” item shows how many servers are currently running on the node/cluster. Changes in this count are indicated by the second, parenthesized number.

2. **General Server Counters**
   The server counters keep track of the total connections received, accepted, rejected, etc., totals for each request type (file transfer, directory listing, image mapping, etc.).

3. **Proxy Serving Counters**
   The server counters keep track of proxy serving connections, network and cache traffic, cache status, etc.

4. **Latest Request**
   This section provides the response status code, and some transaction statistics, the service being accessed, originating host and HTTP request. Note that long request strings may be truncated (indicated by a bolded ellipsis).

5. **Status Message**
   If the server is in an exceptional condition, for example exited after a fatal error, starting up, etc., a textual message may be displayed in place of the the request information. This may be used to initiate remedial actions, etc.

The following shows example output:

```
   I64::           HTTPDMON v2.4.0 IA64          Tuesday, 17-NOV-2009 23:57:25

Process: WASD:80  PID: 24400436  User: HTTP$SERVER  Version: 10.0.0
     Up: 1 05:23:30.67  CPU: 0 00:00:02.05  Startup: 2  Exit: %X00000001
Pg.Flts: 2269  Pg.Used: 8%  WsSize: 36640  WsPeak: 16864
    AST: 1996/2000  BIO: 1998/2000  BYT: 1992640/1992640  DIO: 1000/1000
    ENQ:  477/500   FIL:  296/300   PRC:     100/100       TQ:   98/100

Request: 107  Current: 0/0  Throttle: 0/0/0%  Peak: 4/2
 Accept: 44  Reject: 0  Busy: 0  SSL: 35/80%
CONNECT: 0  GET: 107  HEAD: 0  POST: 0  PUT: 0  (0)
  Admin: 80  Cache: 9/0/0  DECnet: 0/0  Dir: 1
    DCL: CGI:1 CGIplus:1/0 RTE:0/0 Prc:2/0
   File: 16/6  Proxy: 0  Put: 0  SSI: 0  WebDAV: 0/0

    0xx: 0  2xx: 93  3xx: 6  4xx: 7 (403:1)  5xx: 0
     Rx: 67,162 (6 err) Tx: 939,153 (0 err) (3.9kB/S)

   Time: 17 23:56:52  Status: 200  Rx: 835  Tx: 14,762  Dur: 0.459 (34.0kB/S)
Service: https://i64.kednos.com:443
   Host: clive.vsm.com.au (150.101.13.12)
Request: GET /httpd/-/admin/ (********.***)
```

The "/HELP" qualifier provides a brief usage summary.

The server counter values are carried over when a server (re)starts (provided the system has stayed up). To reset the counters use the online Server Administration facility (Chapter 9).

If [DNSlookup] is disabled for the HTTP server the HTTPDMON utility attempts to resolve the literal address into a host name. This may be disabled using the /NORESOLVE qualifier.

13.10 MD5digest

From RFC1321 . . .

" The [MD5] algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. "

The MD5DIGEST utility is primarily provided with WASD for verifying kits as unchanged from the originals released. With the proliferation of mirror sites and other distribution resources it has become good practice to ensure kits remain unchanged from release, to distribution, to installation site (changes due to to data corruption or malicious intent - as remote a possibility as that may seem). Of course it may also be used for any other purpose where the MD5 hash is useful.

For verifying the contents of a WASD release connect to the **original** WASD distribution site, refer to the download page, and make a comparison between the release MD5 hash found against the list of all archive hashes and the MD5 hash of your archive. That can be done as follows

```
$ MD5DIGEST == "$WASD_EXE:MD5DIGEST"
$ MD5DIGEST device:[dir]archive.ZIP
```

The result will look similar to

```
MD5 (kits:[000000]htroot710.zip;1) = 404bbdfe0f847c597b034feef2d13d2d
```

Of course, if you have not yet installed your first WASD distribution using the MD5DIGEST utility that is part of it is not feasable. The original site can provide kits and pre-built executables for this purpose.

13.11 QDLogStats

Quick-and-Dirty LOG STATisticS is a utility to extract very elementary statistics from Web server common/combined format log files. It is intended for those moments when we think "I wonder how many times that new archive has been downloaded?", "How much data was transfered during November?", "How often is *such-and-such* a client using the authenticated *so-and-so* service?", "How much has the mail service been used?" . . . and want the results in a matter of seconds (or at least a few tens of seconds ;-) It is available at the command-line and as a CGI script.

online graphic

For QDLOGSTATS to be available as a CGI script it **must** have authorization enabled against it (to prevent potential ad hoc browsing of a site's logs). The following provides some indication of this configuration, although of course it requires tailoring for any given site.

```
[VMS]
/cgi-bin/qdlogstats ~webadmin,131.185.250.*,r+w ;
```

It could then be accessed using

```
http://the.host.name/cgi-bin/qdlogstats
```

The initial access provides a form allowing the various filters and other behaviours to be selected. The CGI form basically parallels the command-line behaviour described below.

Filters

A number of filters allow subsets of the log contents to be selected. These filters support the same string matching expressions as the server (see "WASD Web Services - Install and Config").

A knowlege of the format and contents of the *common* and *combined* log formats will assist in deciding which and to what purpose filters should be used. Record filtering is done in the same order as is finally displayed, so *method* would be processed before *user-agent* for instance. Normally a record match terminates on the first non-matched filter (to expedite processing). To compare and report each filter for every record apply the /ALL qualifier. To view records as they are processed use the /VIEW qualifier. This by default displays all matched records, but the optional =ALL or =NOMATCH parameters will display all records, or all those but the matches.

$ QDLOGSTATS log-file-spec [pattern qualifiers] [other qualifiers]

Parameters and Qualifiers

Parameter	Description
/ALL	compare and report on all supplied filters
/AUTHUSER=	pattern (any authenticated username)
/BEFORE=	log files before this VMS date/time
/CLIENT=	pattern (client host name or IP address)
/DATETIME=	pattern (“11/Jun/1999:14:08:49 +0930”)
/DECODE[=*keyword*]	URL-decode PATH, QUERY, REFERER before match
/METHOD=	pattern (HTTP “GET”, “POST”, etc.)
/OUTPUT=	file specification
/PATH=	pattern (URL path component only)
/PROGRESS	show progress during processing; a “+” for each file started, a “.” for each 1000 records processed
/QUERY=	pattern (URL query component only)
/REFERER=	pattern (HTTP “Referer:” field, COMBINED only)
/REMOTEID=	pattern (RFC819 file)
/RESPONSE=	pattern (HTTP response code)
/SINCE=	log files after this VMS date/time
/SIZE[=*keyword*]	response size (in bytes) MIN=*integer* MAX=*integer*
/USERAGENT=	pattern (HTTP “User-Agent:” field, COMBINED only)
/VIEW[=type]	display matching log records (ALL, NOMATCH, MATCH)

Usage Examples

- Records from September 1999.

```
$ QDLOGSTATS WASD_LOGS:*1999*.LOG /DATE="*/SEP/1999*"
```

- Records where the browser was an X-based Netscape Navigator

```
$ QDLOGSTATS WASD_LOGS:*.LOG /USERAGENT=*MOZILLA*X11*
```

- Records of POST method requests

```
$ QDLOGSTATS WASD_LOGS:*.LOG /METHOD=POST
```

- Records requesting a particular path

```
$ QDLOGSTATS WASD_LOGS:*.LOG /PATH="/cgi-bin/*"
```

- Select proxy records requesting (a) particular site(s)

```
$ QDLOGSTATS WASD_LOGS:*8080*.LOG /PATH="http://*.compaq.com*"
$ QDLOGSTATS WASD_LOGS:*8080*.LOG /METHOD=POST /PATH="http://*sex*.*/*" /VIEW
```

- Records where the request was authenticated

```
$ QDLOGSTATS WASD_LOGS:*.LOG /AUTHUSER=DANIEL
```

13.12 SECHAN Utility

The SECHAN utility (pronounced “session”) is used by [INSTALL]SECURE.COM and associated procedures to make file system security settings. It is also available for direct use by the site administrator. See “WASD Web Services - Install and Config” .

13.13 StreamLF Utility

This simple procedure used the FDL facility to convert files to STREAM_LF format. The WASD HTTPd server access STREAM_LF files in block/IO-mode, far more efficiently that the record-mode required by variable-record format files.

NOTE: The server can also be configured to automatically convert any VARIABLE record format files it encounters to STREAM_LF.

13.14 WASDbench :^)

WASDbench - an analogue to ApacheBench (Section 13.6) Why have it? ApacheBench only compiles and runs on VMS 7.n and later. This version should compile and run for all supported WASD configurations. It also has the significant performance advantage (looks like ~25%) of using the underlying $QIO services and not the socket API, and is AST event driven rather than using the likes of select( ). It is not a full implementation of AB (for instance, it currently does not do POSTs). The CLI attempts to allow the same syntax as used by AB (within the constraint that not all options are supported) so that it is relatively easy to switch between the two (perhaps for comparison purposes) if desired.

The following examples illustrate its use.

```
$ WB -H
$ WB -C 10 -N 100 http://the.server.name/wasd_root/exercise/0k.txt
$ WB -C 50 -N 500 -K http://the.server.name/wasd_root/exercise/64k.txt
$ WB -C 10 -N 100 http://the.server.name/cgi-bin/cgi_symbols
```

WASDbench also has an *exercise* option, functionality is not found in ApacheBench. It is basically to supercede similar functionality provided by the retired WWWRKOUT. The exercise functionality allows WASDbench to be used to stress-test a server. This behaviour includes mixing HEAD (~5%) with GET requests, and breaking requests during both request and response transfers (~5%). These are designed to shake up the server with indeterminate request types and client error behaviours. The best way to utilize this stress-testing is wrap WASDbench with a DCL procedure providing a variety of different requests types, quantities and concurrencies.

```
$!(example "wrapper" procedure)
$ IF P1 .EQS. "" THEN P1 = F$GETSYI("NODENAME")
$ WB = "$WASD_EXE:WB"
$ SPAWN/NOWAIT WB +e +s +n -n 100 -c  5    http://'p1'/wasd_root/exercise/0k.txt
$ SPAWN/NOWAIT WB +e +s -k -n  50 -c  5 -k http://'p1'/wasd_root/exercise/64k.txt
$ SPAWN/NOWAIT WB +e +s    -n  50 -c  2    http://'p1'/cgi-bin/conan
$!(delay spawning anymore until this one concludes)
$              WB +e +s    -n 100 -c  5    http://'p1'/wasd_root/*.*
$ SPAWN/NOWAIT WB +e +s +n -n 100 -c  1    http://'p1'/wasd_root/exercise/16k.txt
$ SPAWN/NOWAIT WB +e +s    -n  10 -c  1    http://'p1'/cgi-bin/doesnt-exist
$ SPAWN/NOWAIT WB +e +s -k -n  50 -c  2    http://'p1'/cgi-bin/conan/search
$!(delay spawning anymore until this one concludes)
$              WB +e +s    -n  50 -c  2    http://'p1'/wasd_root/src/httpd/*.*
$!(etc.)
```

13.15 WOTSUP Utility

The "WASD Over-The-Shoulder Uptime Picket" is designed to monitor WASD in a production environment for the purpose of alerting operations staff to conditions which might cause that production to be adversely impacted.

Alert triggers include:

- server image exit and/or startup (default)
- server process non-existent or suspended (default)
- percentage thresholds on process quotas (optional)
- rates of HTTP status counter change (optional)
- maximum period without request processing (optional)

Alert reports can be delivered via any combination of:

- OPCOM message
- MAIL
- site-specific DCL command executed in a spawned subprocess
- log file entry

The utility runs in a detached process and monitors the server environment by periodically polling various server data at a default interval is 15 seconds. As the utility requires access to global memory accounting a per-system WOTSUP is required for each node to be monitored.

The following (somewhat contrived) example illustrates the format and content of a WOTSUP report delivered via OPCOM. Reports delivered via other mechanisms have the same content and similar format.

```
%%%%%%%%%%  WOTSUP  24-OCT-2006 13:32:56.44  %%%%%%%%%%
Message from user SYSTEM on KLAATU
Over-The-Shoulder (WASD_WOTSUP) reports:
1. server PID 001C0950 exit %X00000001 (%SYSTEM-S-NORMAL)
2. server STARTUP (10)
3. server PIDs are 0018C14F (HTTPd:80), 001C0950 (HTTPe:80)
4. pagfilcnt:395432 pgflquota:500000 79% <= 80%
```

For further infomation check the descriptive prologue in the WASD_ROOT:[SRC.UTILS]WOTSUP.C source code.