WASD Hypertext Services - Technical Overview

June 2004

For version 8.5 release of the WASD VMS Hypertext Services HTTP server.

Supercedes:

January 2004 (v8.4)
July 2003 (v8.3)
April 2003 (v8.2)
December 2002 (v8.1)
July 2002 (v8.0)
July 2001 (v7.2)
November 2000 (v7.1)
June 2000 (v7.0)
December 1999 (v6.1)
May 1999 (v6.0)
November 1998 (v5.3)
September 1998 (v5.2)
July 1998 (v5.1)
March 1998 (v5.0)
November 1997 (v4.5)
October 1997 (v4.4)
August 1997 (v4.3)
July 1997 (v4.2)
June 1997 (v4.1)
October 1996 (v3.4)
December 1995 (initial *freeware* release, v3.1)
August 1995 (v2.3)
May 1995 (v2.1)

Abstract

This document is a technical overview of the implementation of the WASD VMS HyperText Transport Daemon. It contains information on server startup, configuration, etc. Previous versions contained information on scripting and brief descriptions of the major code modules of the server, these have now been removed to a documents of their own (see below).

Also see “WASD Scripting Environment” for information on CGI, ISAP, OSU, etc., scripting, and “WASD Hypertext Environment” for information on using the WASD VMS Hypertext Services.

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

Author

Mark G. Daniel
Intelligence, Surveillance & Reconnaissance Division
Defence Science and Technology Organisation

For WASD-related email please use Mark.Daniel@wasd.vsm.com.au

Should the above address present problems or provide no response for an extended period then use Mark.Daniel@dsto.defence.gov.au

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

+61 (8) 82596189 (bus)
+61 (8) 82596673 (fax)

PO Box 1500
Edinburgh
South Australia 5108

Online Search

online search

Online PostScript

This book is available in PostScript for access by suitable viewers (e.g. GhostScript). Use a browser to access a print menu in this same location. It can be reprinted from the location HT_ROOT:[DOC.HTD]HTD.PS

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 Hypertext Services

Copyright © 1996-2004 Mark G. Daniel.

This package 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; version 2 of the License, or any later version.

This package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

See HT_ROOT:[000000]GNU_GENERAL_PUBLIC_LICENSE.TXT

You should have received a copy of the GNU General Public License along with this package; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The Apache Group

This product includes software developed by the Apache Group for use in the Apache HTTP server project (http://www.apache.org/).

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

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 ...
```

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.
```

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.
```

Bailey Brown Jr.

LZW compression is implemented using code derived in part from the PBM suite. This code is copyright by the original author:

```
* GIF Image compression - LZW algorithm implemented with Tree type
*                         structure.
*                         Written by Bailey Brown, Jr.
*                         last change May 24, 1990
*                         file: compgif.c
*
*  You may use or modify this code as you wish, as long as you mention
*  my name in your documentation.
```

Other

OpenVMS, **Compaq TCP/IP Services for OpenVMS**, **Compaq C**, **Alpha** and **VAX** are registered trademarks of Hewlett Packard Corporation.

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 HTTPd - Overview

2.1 Server Behaviour 2–3
2.2 VMS Versions 2–4
2.3 TCP/IP Packages 2–4
2.4 International Features 2–4
2.5 HTTP Methods Usage 2–6
2.5.1 GET 2–6
2.5.2 POST & PUT 2–6
2.5.3 DELETE 2–8

Chapter 3 New to WASD? Start Here!

Chapter 4 Installation and Update

4.1 Source Archive, Object Module Archives 4–3
4.2 Accessable Volume 4–4
4.3 ODS-5 Volumes 4–4
4.4 Package Directory Structure 4–4
4.5 Installation DCL Procedure 4–5
4.6 Update DCL Procedure 4–6
4.7 Quick-Check 4–7
4.8 ‘‘Clone’’ Procedure 4–9
4.9 Re-Linking 4–9
4.10 VMS 6.0 and 6.1 4–9
4.11 VMS 5.5-*n* 4–9

4.12 Local Setup Suggestions 4–10
4.13 Reporting Problems 4–10

Chapter 5 Server Account and Environment

5.1 VMS Server Account 5–2
5.2 VMS Scripting Account 5–2
5.3 Account Support Files 5–3
5.4 Other Resources 5–5
5.5 HTTPd Command Line 5–7
5.5.1 Server Startup 5–7
5.5.2 Server Command Line Control 5–10
5.5.2.1 Accounting 5–10
5.5.2.2 Authentication 5–11
5.5.2.3 Cache 5–11
5.5.2.4 DCL/Scripting Subprocesses 5–11
5.5.2.5 DECnet Scripting Connections 5–11
5.5.2.6 Instances 5–12
5.5.2.7 Logging 5–12
5.5.2.8 Mapping 5–12
5.5.2.9 Shutdown and Restart 5–13
5.5.2.10 Secure Sockets Layer 5–13
5.5.2.11 Throttle 5–13

Chapter 6 Server Configuration

6.1 Site Organisation 6–1
6.2 Server Instances 6–3
6.3 Virtual Services 6–5
6.4 Request Throttling 6–8
6.5 Client Concurrency 6–10
6.6 Content-Type Configuration 6–11
6.6.1 Adding Content-Types 6–11
6.6.2 MIME.TYPES 6–11
6.6.3 Unknown Content-Types 6–13
6.6.4 Explicitly Specifying Content-Type 6–14
6.7 Language Variants 6–15
6.8 Character Set Conversion 6–16
6.9 Error Reporting 6–17
6.9.1 Basic and Detailed 6–17
6.9.2 Site Specific 6–18
6.10 OPCOM Logging 6–21

6.11 Access Logging 6–21
6.11.1 Log Format 6–21
6.11.2 Log Per-Period 6–23
6.11.3 Log Per-Service 6–24
6.11.4 Log Per-Instance 6–24
6.11.5 Log Naming 6–25
6.11.6 Access Tracking 6–26
6.11.7 Access Alert 6–26
6.12 Include File Directive 6–27

Chapter 7 "Securing" The Site

7.1 Recommended Package Security 7–2
7.2 Maintaining Package Security 7–4
7.3 Independent Package and Local Resources 7–6
7.4 Configuration 7–6
7.4.1 Directory Listings 7–6
7.4.2 Server Reports 7–7
7.4.3 Scripting 7–7
7.4.4 Server Side Includes 7–7
7.5 Scripting 7–8
7.6 Authorization 7–8
7.7 Miscellaneous Issues 7–9
7.8 Site Attacks 7–10

Chapter 8 Server Configuration Directives

8.1 Functional Groupings 8–1
8.2 Alphabetic Listing 8–7

Chapter 9 Service Configuration

9.1 IPv4 and IPv6 9–2
9.2 Service Directives 9–3
9.3 Directive Detail 9–4
9.4 Administration 9–6
9.5 Examples 9–7

Chapter 10 Message Configuration

10.1 Behaviour 10–1
10.2 Message File Format 10–2
10.3 Supplied Message Files 10–3

Chapter 11 String Matching

11.1 Wildcard Patterns 11–1
11.2 Regular Expressions 11–2
11.3 Examples 11–4
11.4 Expression Substitution 11–4

Chapter 12 Conditional Configuration

12.1 Conditional Syntax 12–2
12.2 Conditional Keywords 12–3
12.3 Examples 12–9

Chapter 13 Mapping Rules

13.1 Rule Interpretation 13–2
13.2 VMS File System Specifications 13–3
13.3 Extended File Specifications (ODS-5) 13–4
13.3.1 Characters In Request Paths 13–4
13.3.2 Characters In Server-Generated Paths 13–5
13.4 Rules 13–5
13.4.1 MAP, PASS, FAIL Rules 13–6
13.4.2 REDIRECT Rule 13–6
13.4.3 USER Rule 13–7
13.4.4 EXEC/UXEC and SCRIPT, Script Mapping Rules 13–8
13.4.5 SET Rule 13–10
13.5 Mapping Examples 13–20
13.6 Virtual Servers 13–21
13.7 Conditional Mapping 13–22
13.8 Mapping User Directories (*tilde* character (‘‘~’’)) 13–26
13.8.1 Using The SYSUAF 13–26
13.8.2 Without Using The SYSUAF 13–27

Chapter 14 Authorization Quick Guide

14.1 SYSUAF/Identifier Authentication 14–1
14.2 SYSUAF Authentication 14–2
14.3 Other Authentication 14–3
14.4 Read and Write Groupings 14–4
14.5 Considerations 14–4

Chapter 15 Authentication and Authorization

15.1 Rule Interpretation 15–2
15.2 Authentication Policy 15–2
15.3 Permissions, Path and User 15–4
15.4 Authorization Configuration File 15–4
15.5 Authorization Sources 15–8
15.6 Realm, Full-Access, Read-Only 15–12
15.7 Virtual Servers 15–13
15.8 Authorization Configuration Examples 15–14
15.8.1 KISS 15–15
15.9 Authorization Cache 15–16
15.10 SYSUAF-Authenticated Users 15–17
15.10.1 ACME 15–17
15.10.2 Rights Identifiers 15–18
15.10.3 WASD ‘‘Hard-Wired’’ Identifiers 15–19
15.10.4 VMS Account Proxying 15–20
15.10.5 Nil-Access VMS Accounts 15–22
15.10.6 SYSUAF and SSL 15–22
15.10.7 SYSUAF Security Profile 15–23
15.10.8 SYSUAF Profile For Full Site Access 15–24
15.11 Skeleton-Key Authentication 15–24
15.12 Controlling Server Write Access 15–25
15.13 Securing All Requests 15–26
15.14 User Password Modification 15–27
15.15 Cancelling Authorization 15–28

Chapter 16 Proxy Services

16.1 HTTP Proxy Serving 16–2
16.1.1 Enabling A Proxy Service 16–3
16.1.2 Proxy Bind 16–3
16.1.3 Proxy Chaining 16–4
16.1.4 Controlling Proxy Serving 16–4

16.2 Caching 16–6
16.2.1 Cache Device 16–7
16.2.2 Enabling Caching 16–8
16.2.3 Cache Management 16–8
16.2.4 Cache Invalidation 16–10
16.2.5 Cache Retention 16–11
16.2.6 Reporting and Maintenance 16–12
16.2.7 PCACHE Utility 16–12
16.3 CONNECT Serving 16–14
16.3.1 Enabling CONNECT Serving 16–14
16.3.2 Controlling CONNECT Serving 16–15
16.4 FTP Proxy Serving 16–15
16.4.1 FTP Query String Keywords 16–16
16.4.2 ‘‘login’’ Keyword 16–17
16.5 Gatewaying Using Proxy 16–17
16.5.1 Reverse Proxy 16–18
16.5.2 One-Shot Proxy 16–19
16.5.3 DNS Wildcard Proxy 16–19
16.5.4 Originating SSL 16–21
16.6 Browser Proxy Configuration 16–22

Chapter 17 Secure Sockets Layer

17.1 SSL Functionality Sources 17–3
17.2 WASD SSL Quick-Start 17–4
17.3 SSL Configuration 17–5
17.3.1 HTTPD$CONFIG [Service] 17–5
17.3.2 HTTPD$SERVICE 17–6
17.3.3 SSL Server Certificate 17–6
17.3.4 SSL Private key 17–6
17.3.5 SSL Virtual Services 17–7
17.3.6 SSL Access Control 17–7
17.3.7 Authorization Using X.509 Certification 17–7
17.3.8 Features 17–8
17.3.9 X509 Configuration 17–9
17.3.10 Certificate Authority Verification File 17–12
17.3.11 X.509 Authorization CGI Variables 17–13
17.4 Certificate Management 17–15
17.4.1 Server Certificate 17–16
17.4.2 Client Certificate 17–17
17.4.3 Certificate Signing Request 17–18
17.5 SSL CGI Variables 17–21

17.6 SSL References 17–22

Chapter 18 Server Administration

18.1 Access Without Configuration 18–1
18.2 Access Configuration 18–2
18.3 Server Instances 18–3
18.4 HTTPd Server Reports 18–4
18.5 HTTPd Server Revise 18–7
18.6 HTTPd Server Action 18–9

Chapter 19 WATCH Facility

19.1 Server Instances 19–2
19.2 Event Categories 19–2
19.3 Request Filtering 19–5
19.4 Report Format 19–6
19.5 Usage Suggestions 19–7
19.6 Command-Line Use 19–8

Chapter 20 Cache

20.1 Non-File Content Caching 20–2
20.2 Permanent and Volatile 20–3
20.3 Cache Suitability Considerations 20–3
20.4 Cache Content Validation 20–5
20.5 Cache Configuration 20–5
20.6 Cache Control 20–7
20.7 Circumventing The Cache 20–7

Chapter 21 Server Performance

21.1 Simple File Request Turn-Around 21–2
21.2 Scripting 21–4
21.3 SSL 21–7
21.4 Suggestions 21–7

Chapter 22 HTTPd Web Update

Chapter 23 Utilities and Facilities

23.1 Echo Facility 23–1
23.2 Hiss Facility 23–2
23.3 Where Facility 23–2
23.4 Xray Facility 23–2
23.5 Apache Bench 23–2
23.6 CALogs 23–3
23.7 HTAdmin 23–4
23.8 HTTPd Monitor 23–6
23.9 MD5digest 23–7
23.10 QDLogStats 23–8
23.11 SECHAN Utility 23–9
23.12 Scrunch Utility *(obsolete)* 23–10
23.13 StreamLF Utility 23–10
23.14 WASD Bench :^) 23–10
23.15 Server Workout *(obsolete)* 23–11

Chapter 1
Introduction

This document provides an technical overview of the WASD VMS HTTP server. It is primarily written for use internal to WASD and assumes that perpsective without apology. Any additional usage is subordinate to its role within WASD. The software has been written expressly for supporting an intra-organisational hypertext environment on a VMS platform. It too, is unreservedly tailored to this purpose and the requirements of WASD. All programs were designed only to specifically comply with the requirements of Compaq C, within a Compaq TCP/IP Services for VMS environment.

The document **assumes** a basic understanding of the hypertext 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 (these are often best consulted online, on the Internet WWW).

Also see ''WASD Scripting Environment'' for information on using CGI, ISAPI, OSU, etc. scripting, and ''WASD Hypertext Environment'' for information on using the WASD VMS hypertext facilities.

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

Objectives

The primary impetus for an internal Web environment was a 1993 decision by Division management to make as much information as possible, both administrative and research, available online (to use the current term . . . an *intranet*). Early experimentation with a *Gopher* implementation soon made way for the obvious advantages of the emerging Web technology.

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 a Local HTTPd

Reasons for developing a local HTTP server are few but compelling:

- The WASD (then HFRD) Web implementation began mid-1994 (October 1997 note).
- It was prefered to support the hypertext environment on a VMS platform. This is currently (1995) the most widely used and accessable environment within WASD.
- Existing servers (and there are quite a few variations) are largely Unix based, although it is being supported (to a greater or lesses extent) across a wide range of platforms. Ports to VMS, if they exist, are 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 is not multi-threaded under VMS (i.e. cannot support concurrent clients). For example, a lengthy search may delay other clients for unacceptable periods.
  - — Its 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 (DECthreads) server was evaluated via documentation mid-1994. (now in 1998 . . . I am reticent to make further comment on now such a widely deployed package :^)
- HyperText Transport Protocol, in its current form, is relatively simple to implement to the level required to support intra-Divisional requirements.
- **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.
- **As of June 1997** the server and its 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.
- **As of 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.
- **As of July 2002** it continues to be refined and extended. A greater emphasis on ''commercial'' functionality has occured over the past couple of years.

Chapter 2
HTTPd - Overview

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

General

- concurrent, multi-threaded client support
- ‘‘GET’’, ‘‘HEAD’’, ‘‘POST’’ and ‘‘PUT’’ HTTP method support
- virtual services (servers)
- concurrent 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, IA64 and (now - 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 (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 caching for all HTTP/1.0 methods, plus the CONNECT method (allowing SSL tunnelling), along with FTP proxy (non-caching)
- gatewaying between protocols (HTTP-to-SSL, SSL-to-HTTP, HTTP-to-FTP, etc.)

- clickable-image support (both NCSA and CERN formats)

Scripting

- CGI-compliant scripting
- 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
- ‘‘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) 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)
- RFC1413 (*ident* daemon) ‘‘authentication’’

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 HTTPd executes permanently on the server host, listening for client connection requests on TCP/IP port 80 (by default). It provides concurrent services for a (technically) unlimitted number of clients (constrained only by the server resources). When a client connects the server performs the following tasks:

1. creates a *thread* for this request
2. reads and analyzes the HTTP request sent, depending on the nature of the request . . .
   - transfer of a file (possibly from cache)
   - processing of an SSI HTML file
   - directory listing
   - file/directory create/update
   - server administration
   - processing of a clickable-image mapping file
   - create a process to to provide a CGI scripting environment:
     — standard CGI
     — WASD CGIplus
     — WASD Run-Time Environment
   - connects to a DECnet object to provide a scripting environment:
     — standard CGI
     — OSU (DECthreads) emulated
3. closes the connection to the client and disposes of the thread data structures

For I/O intensive activities like file transfer and directory listing, the AST-driven code provides an efficient, multi-threaded environment for the concurrent serving of multiple clients.

For scripts, the technique of using multi-threaded, concurrent, spawned subprocesses, detached autonomous processes, or network processes, attached to standard input/output streams, provides a versatile, extensible, powerful scripting environment. Any DCL procedure or image executing within the process can behave essentially as an HTTP server. This capability is employed to easily extend the basic services provided by the core daemon code. An HTTP script/server for this environment does not need to concern itself with network activities, it merely reads and writes from the standard I/O streams.

2.2 VMS Versions

The WASD server is officially supported on any VMS version from V6.0 upwards, on Alpha, IA64 (Itanium) and VAX architectures. Obviously no guarantees can be made for yet-to-be-released versions but at a worst-case these should only require relinking. Pre 7.*n* versions of WASD have also been known to compile and run successfully under V5.5-*n* (Section 4.11).

WASD on IA64 based systems has been initially ported and tested against ''HP OpenVMS Industry Standard 64 Evaluation Release Version 8.1''. Later VMS releases on this platform may require recompilation. No significant code changes were required for this port (as might be expected), mainly requirements for Alpha specific conditional compilation expanded to include IA64.

Non-server account scripting requires a minimum VMS V6.2, to provide the $PERSONA services required for this functionality. Equivalent functionality on earlier versions of VAX VMS (i.e. 6.0 and 6.1) is available using the PERSONA_MACRO build option (Section 4.10).

The WASD distribution and package organisation fully supports mixed-architecture clusters (Alpha, IA64 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.

- Compaq 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 (needless-to-say).

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 Section 6.7.

- **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 Chapter 8.

- **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 Chapter 10.

- **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 Chapter 6.

- **Conditional Rule Mapping**

Mapping rules map requested URL paths to physical or other paths (Chapter 13). 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 Section 13.7.

2.5 HTTP Methods Usage

This section describes WASD-specific characteristics of the available HTTP/1.0 request methods.

2.5.1 GET

Of course, the GET method is used to access documents supplied by the server. There is nothing WASD server-specific about this method.

2.5.2 POST & PUT

The WASD HTTPd does not differentiate between POST and PUT methods, both are handled identically.

Script Handling

The ‘‘normal’’ usage of the POST method is to return data from a <FORM>..</FORM> construct to a script running on the server. In this regard WASD is no different to other any web server; the form data is delivered to the script's standard input as a stream of URL-encoded text. For example:

```
name=Fred+Nurk&address=Fred%27s+House%0D%0A0+Nowhere+Lane&submit=Submit
```

Note that WWW_CONTENT_LENGTH will be the length of the form data. See ‘‘Scripting Environment’’ document for further information.

File Creation/Upload

If the client sends data back to the server using either the POST or the PUT methods, without a script being mapped to be executed in response to that data, the WASD HTTPd will create a file corresponding to the specified path. The data stream may be text or binary.

Of course, for the server to accept POST and PUT data in this manner, authentication and authorization must be enabled and allow such access to the request path.

The data stream is processed according to MIME content-type:

- **application/x-www-form-urlencoded**

  The server specially processes ‘‘application/x-www-form-urlencoded’’ POSTS (i.e. those generated by <FORM>...</FORM>, allowing files to be created directly from HTML forms. The processing eliminates any field names from the URL-encoded data stream, placing

only field values into the file. This capability can be quite useful and is demonstrated in the Update HTTPd module

online hypertext link.

- **multipart/form-data**

  This server can process a request body according to RFC-1867, ''Form-based File Upload in HTML''. As yet it is not a full implementation. It will not process "multipart/mixed" subsections. The implementation is basic, providing a facility to allow the upload of a file into the server administered file system. The ACTION= parameter of the <FORM> tag must specify the directory (as a URL path) in which the uploaded file will be created.

  The following example HTML illustrates how a form may be used to upload a file from the browser host file system:

```
<FORM METHOD=POST ACTION="/web/directory/" ENCTYPE="multipart/form-data">
<INPUT TYPE=submit VALUE=" Upload document ... ">
<INPUT TYPE=file SIZE=50 NAME=uploadfile>
</FORM>
```

  online hypertext link

  Note

  This capability has only been tested against Netscape Navigator versions 2 and 3. VMS Netscape Navigator 3.0b5 hangs if an upload of a variable-record format file is attempted. Stick to STREAM-LF or fixed, or convert the file to STREAM-LF.

- **text/html**
  text/plain

  Text file created according to the path, VMS record type is STREAM-LF.

- **image/gif**
  application/octet-stream
  etc., etc., etc.

  Any other MIME type is considered *binary* and the created file is made an UNDEFINED record type.

Directory Creation

A directory will be created by the HTTPd if a directory path is provided with the POST or PUT methods. For example:

```
/dir1/dir2/dir-to-be-created/
```

File Deletion

A file will be deleted by the HTTPd if the file path ending with a wildard version specification is provided with the POST or PUT methods. For example:

```
/dir1/dir2/file-to-be.deleted;*
```

Directory Deletion

A directory will be deleted by the HTTPd if a directory path ending with a wildard version specification is provided with the POST or PUT methods. For example:

```
/dir1/dir2/dir-to-be-deleted/;*
```

2.5.3 DELETE

The DELETE method should delete the file or directory corresponding to the supplied path.

Chapter 3

New to WASD? Start Here!

This chapter provides a quick guide to getting your WASD package installed, configured and serving. This covers initial installation.

1. **Unzip Package**

   Install the files following the guidelines in Chapter 4. **Note** that more than one archive may be needed (Section 4.1).

2. **INSTALL Package**

   Server installation is performed using the INSTALL.COM procedure (Section 4.5).

   - **Build Package -** Compile and link, or just link supplied object files to produce VMS executables for the system's version of VMS.
   - **Check Package -** Check basic operation of the package (Section 4.7).
   - **Create Server and Scripting Accounts -** Create two independent accounts, one for executing the server, the other for executing scripts (Section 5.1). If quotas are enabled on the target disk provides an ambit allocation for these accounts. Review this at some stage.
   - **Set Package Security -** This sections traverses the newly installed tree and sets all package directories and files to required levels of access (Section 7.2).
   - **Copy Support and Configuration Files -** Copy the example server support and configuration files (Section 5.3).
   - **Install Scripts -** Selectively copy groups of scripts from package build directories into the scripting directories.

3. **Configure Package**

   Following the execution of the INSTALL.COM procedure the package should require only minor, further configuration.

Initially two files may require alteration.

1. The startup file, possibly to set the local HTTPD$GMT logical. Consider using the STARTUP_LOCAL.COM file for other site-specific requirements (Section 5.3).
2. The only configuration that should require immediate attention will be to the mapping rules (Chapter 13).

More generally server runtime configuration involves the considerations discussed in Section 6.1 along with the following aspects:

- Configuring the HTTP server run-time characteristics (Chapter 6).
- Mapping request paths to the VMS file system, and to other things such as scripts (Chapter 13).
- Customizing some messages (or all if non-English language environment) (Chapter 10).
- Establishing an authentication and authorization environment (Chapter 15).

4. **Start Server**

   Execute the startup procedure. Get your browser and connect!

5. **Find Out What's Wrong :^(**

   Of course *something* will not be right! This can happen with the initial configuration and sometimes when changing configuration. The server provides information messages in it's run-time log, look in the HT_ROOT:[LOG_SERVER] directory.

   Remember, the basic installation's integrity can always be checked as described in Chapter 4, Section 4.7. This uses the configuration files from the [EXAMPLE] directory, so provided these have not been altered the server should execute in *demonstration mode* correctly.

   Can't resolve it? See Section 4.13.

Chapter 4
Installation and Update

The WASD package and updates will always be distributed as ZIP archives.

It generally pays to use the latest version of VMS UNZIP available. Archives will contain a comment about the minimum version required, check that as described in the next paragraph. To show the version of the current UNZIP utility, use

```
$ UNZIP -v
```

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
```

It is recommended to check the integrity of, then list the contents of, the archive before UNZIPping.

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

Paranoid?

Are you sure you got what you thought you did? Not keen on importing even one trojan into your site? You may wish to confirm the archive you are about to install is what was originally released. The MD5DIGEST utility may ease your (now) troubled mind. See Section 23.9.

Installation UNZIP

For complete package distributions the archive will contain the complete directory tree. Hence to install or do a complete update it is necessary to SET DEFAULT into the top-level directory of the disk the package is to be installed on.

```
$ SET DEFAULT device:[000000]
```

A complete installation/update will have the structure:

```
Archive:  HT_ROOT:[000000]HTROOT430.ZIP;1

 WASD (HFRD) VMS Hypertext Services, Copyright (c) 1996,1997 Mark G.Daniel.
 This package (all associated programs), comes with ABSOLUTELY NO WARRANTY.
 This is free software, and you are welcome to redistribute it
 under the conditions of the GNU GENERAL PUBLIC LICENSE, version 2.

  *  Complete v4.3.0 package.
  *  $ SET DEFAULT device:[000000]
  *  $ UNZIP "-V" device:[dir]HTROOT430.ZIP

 VMS file attributes saved ... use UnZip 5.n+ on OpenVMS

 Archive created 1-AUG-1997

 Length    Date    Time    Name
 ------    ----    ----    ----
      0  07-30-97  14:15   ht_root/$_read_1st/
      0  07-30-97  14:15   ht_root/aacrt060/
      0  07-30-97  14:15   ht_root/auth/
      0  07-30-97  14:15   ht_root/axp/
   5360  07-30-97  13:23   ht_root/changes.html
      0  07-30-97  14:15   ht_root/doc/
.
.
.
  16896  07-30-97  09:23   ht_root/vax/wwwrkout.exe
 ------                    -------
19109754                   996 files
```

Update UNZIP

For updates to portions of the package only the tree below HT_ROOT:[000000] is provided, hence it is necessary to SET DEFAULT to HT_ROOT:[000000] before UNZIPping.

```
$ SET DEFAULT HT_ROOT:[000000]
```

An update will have the structure:

```
Archive:  HT_ROOT:[000000]HTROOT430_UPD_01.ZIP;1

 WASD (HFRD) VMS Hypertext Services, Copyright (c) 1996,1997 Mark G.Daniel.
 This package (all associated programs), comes with ABSOLUTELY NO WARRANTY.
 This is free software, and you are welcome to redistribute it
 under the conditions of the GNU GENERAL PUBLIC LICENSE, version 2.

  *  Mostly minor updates to scripts and utilities for the v4.3.0 package.
  *  A couple nuisance-value-bugs have also been fixed in these.
  *  SDM2HTM can now generate <FRAMESET> documents (see [DOC]*.COM)

 VMS file attributes saved ... use UnZip 5.n+ on OpenVMS

 Archive created 6-AUG-1997
```

```
 Length    Date    Time    Name
 ------    ----    ----    ----
  42957  08-04-97  07:57   src/utils/wwwrkout.c
  59664  08-04-97  07:57   src/utils/obj_axp/wwwrkout.obj
  19674  08-04-97  07:57   src/utils/obj_vax/wwwrkout.obj
  26624  08-04-97  07:57   axp/wwwrkout.exe
.
.
.
  45056  08-05-97  11:02   vax/sdm2htm.exe
    617  08-05-97  09:23   doc/all_docs_framed.com
     44  08-05-97  09:23   doc/all_docs_not_framed.com
    544  08-05-97  09:22   doc/readme.html
 ------                    -------
2773107                    90 files
```

Existing Installations

When installing either a full archive or to a lesser extent an update archive ‘‘over the top’’ of an existing installation consider the following.

- Some *insurance* on the directory tree is recommended in case the update should fail or otherwise be unusable or problematic (of course, this is good advice whenever about to make major changes to anything!) This may be in the format of a regular site backup, special pre-update backup, or special pre-update ZIP archive of the directory tree. The latter two could be accomplished using commands similar to the following:

  ```
  $ BACKUP HT_ROOT:[000000...] location:HTROOT.BCK/SAVE/VERIFY

  $ ZIP "-V" location:HTROOT.ZIP device:[HT_ROOT...]*.*
  $ ZIP "-T" location:HTROOT.ZIP
  ```

  If using ZIP then ensure that a previous version of the target ZIP file does not already exist. If it does then that version is updated, a new version is not created.

- For existing files a new version is created (the first time this is about to occur the UNZIPper requests permission - either ‘‘A’’ for all, or ‘‘y’’ or ‘‘n’’ or a per-file basis).

- It is possible to *selectively* extract portions of a tree if something has become damaged. This would be accomplished by specifying what needs to be extracted from the archive (instead of the default *all*), as illustrated by the following example where only the Alpha object modules are extracted.

  ```
  $ SET DEFAULT device:[000000]
  $ UNZIP "-V" device:[dir]archive-AXP.ZIP ht_root/src/httpd/obj_axp/*.*
  ```

4.1 Source Archive, Object Module Archives

The complete package, source code, documentation, examples, etc., is provided in a single main archive. Installation and other build procedures allow the entire package to be compiled and linked from this if prefered. This requires a later version of DEC C (preferably v5.*n* or greater). VAX C is no longer supported.

In addition, for those unable or not wishing to fully build the distribution, two other platform-specific archives are available, AXP (Alpha) and VAX, containing a complete set of object modules, allowing the package to be built via a link operation only.

If a complete build is planned then only the main archive is required. If a link-only build then an additional archive for each architecture must be UNZIPped as described above. This applies to both full installations and subsequent updates. The archives will be clearly identified with the architecture type, as illustrated in this example.

```
$ UNZIP "-V" device:[dir]archive-AXP.ZIP
$ UNZIP "-V" device:[dir]archive-VAX.ZIP
```

Note

The WASD distribution and package organisation fully supports mixed-architecture clusters (AXP, IA64 and/or VAX in the one cluster) as one integrated installation.

4.2 Accessable Volume

Unlikely as it might be to install the package on a private or otherwise protected volume, the server and scripting accounts being unprivileged in themselves, require access sufficient to read, write and delete files from the volume (disk). The following illustrates how to check this and what the protections should look like. Generally any device that an unprivileged user can use the server accounts can use.

```
$ SHOW SECURITY /CLASS=VOLUME $1$DKA0:

ALPHASYS object of class VOLUME
     Owner: [1,1]
     Protection: (System: RWCD, Owner: RWCD, Group: RWCD, World: RWCD)
     Access Control List: <empty>
```

4.3 ODS-5 Volumes

The WASD package has been successfully installed on and used from ODS-5 (extended file specification) volumes. Note that the installation procedures and file system organisation of the package tree has been designed for ODS-2 compliance. As long as this is maintained there should be no issues with actually having it resident on an extended file system compliant volume. (Of course the issue of installing WASD on an ODS-5 volume is completely separate from it's ability to serve the contents of an ODS-5 volume!)

4.4 Package Directory Structure

The package directories and content are organised as follows. Note that only some of these can be accessed by the server (and therefore seen in server-generated directory listings) due to directory and file protections (Section 7.1).

Package Directory Structure

Directory	Description
[AXP-BIN]	Alpha executable script files
[AXP]	Alpha build and utility area

Directory	Description
[CGI-BIN]	architecture-neutral script files
[DOC]	package documentation
[EXAMPLE]	package examples
[EXERCISE]	package test files
[HTTP$NOBODY]	scripting account default home area
[HTTP$SERVER]	server account default home area
[IA64-BIN]	IA64 executable script files
[IA64]	IA64 build and utility area
[INSTALL]	installation, update and security procedures
[LOCAL]	site configuration files
[LOG]	site access logs
[LOG_SERVER]	server process (SYS$OUTPUT) logs
[RUNTIME]	graphics, help files, etc.
[SCRATCH]	working file space for scripts
[SCRIPT]	example architecture-neutral scripts
[SRC]	package source files
[STARTUP]	package startup procedures
[VAX-BIN]	VAX executable script files
[VAX]	VAX build and utility area

4.5 Installation DCL Procedure

The INSTALL.COM procedure assists with the first installation of WASD. It provides a *vanilla* setup, using the standard directories and account environment described in this document. All sections prompt before performing any action and generally default to ''no''. Read the information and questions carefully!

After UNZIPping the package do the following:

```
$ SET DEFAULT device:[HT_ROOT]
$ @INSTALL
```

It performs the following tasks:

1. **Build Executables -** Either compile sources and link, or just link package object code to produce images for the local version of VMS. If the system has a suitable SSL toolkit the installer is requested whether an SSL enabled version be built.
2. **Check Package -** Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package (Section 4.7).

3. **Create Server and Scripting Accounts -** Create two, independent accounts, one for executing the server, the other for executing scripts (Section 5.1). If quotas are enabled on the target disk provides an ambit allocation for these accounts. Review this at some stage.
4. **Set Package Security -** This section traverses the newly installed tree and sets all package directories and files to required levels of access. For directory settings see Section 7.1.
5. **Copy Support and Configuration Files -** Copy the example server support and configuration files (Section 5.3).
6. **Install Scripts -** Selectively copy groups of scripts from package build directories into the scripting directories.

Support files to consider when customizing startup, etc. (see Section 5.3 for further detail):

STARTUP.COM
STARTUP_LOCAL.COM
STARTUP_SERVER.COM

4.6 Update DCL Procedure

The UPDATE.COM procedure assists with subsequent updates of WASD. It assumes a *vanilla* setup, using the standard directories and account environment described in this document. All sections prompt before performing any action and generally default to ''no''. Read the questions carefully!

Of course it is best (read mandatory) for the server to be shut down during an update!

```
$ HTTPD/DO=EXIT/ALL
```

After UNZIPping the updated package do the following:

```
$ @UPDATE
```

It provides the following functions:

1. **Build Executables -** Either compile sources and link, or just link package object code to produce images for the local version of VMS. If the system has a suitable SSL toolkit the installer is requested whether an SSL enabled version be built.
2. **Server Quick-Check -** Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package (Section 4.7).
3. **Server Support/Configuration Files -** Copies changed example HTTP server configuration and support files from the [EXAMPLE] directory to the [HTTP$SERVER], [LOCAL] and [STARTUP] directories.
4. **Update Scripts -** Selectively copy groups of scripts from package build directories into the scripting directories.
5. **Reapply Package Security -** This section traverses the updated tree and sets all package directories and files to required levels of access. For directory settings see Section 7.1.

6. **Post-Update Cleanup -** Prompts for permission to execute the post-update procedure described below.

7. **Purge Files -** Prompts for permission to purge the entire HT_ROOT:[000000...] tree.

If declined during the update procedure the post-update steps 6 and 7 can be performed at any subsequent time using

```
$ SET DEFAULT HT_ROOT:[000000]
$ @UPDATE CLEANUP
$ PURGE [...]
```

4.7 Quick-Check

Once installed or updated it is possible to check the basic package at any time using the [INSTALL]DEMO.COM procedure. This invokes the server image using the /DEMO qualifier allowing some behaviours not possible under general use. Follow the displayed instructions. Basically, the server should start and become reachable via port number 7080. So, to test its availability, using your prefered browser enter the URL listed on line starting with ‘‘%HTTPD-I-SERVICE’’ and the WASD welcome page should be displayed.

```
$ @HT_ROOT:[INSTALL]DEMO.COM

                        ******************************
                        *  WASD PACKAGE DEMONSTRATOR  *
                        ******************************

If you have the SSL package then just add "SSL" as parameter 1!

When finished using demonstrator abort server execution using control-Y
(a subprocess will be spawned to preserve current process environment)

Use a browser to access the "%HTTPD-I-SERVICE" shown when the server starts.

The server will be running in promiscuous mode!
Any username with the password specified below can be used for authentication.
Enter a string to use as a password when later prompted by your browser.

Password (for demo authentication)? []: anyoldpassword
```

```
%DCL-S-SPAWNED, process SYSTEM_24 spawned
%DCL-S-ATTACHED, terminal now attached to process SYSTEM_24
%HTTPD-I-SOFTWAREID, HTTPd-WASD/8.5.0 OpenVMS/AXP
WASD VMS Hypertext Services, Copyright (C) 1996-2004 Mark G.Daniel.
This package (all associated programs), comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under the conditions of the GNU GENERAL PUBLIC LICENSE, version 2.
%HTTPD-I-STARTUP, 02-JAN-2004 21:53:09
%HTTPD-I-SYSTEM, Digital AlphaStation 500/333 VMS V7.3-2
%HTTPD-W-SYSPRV, operating with implicit SYSPRV (UIC group 1)
%HTTPD-I-TCPIP, Compaq TCPIP$IPC_SHR V5.4-15 (18-SEP-2003 21:58:05.32)
%HTTPD-I-MODE, INTERACTIVE
%HTTPD-I-ODS5, supported by Alpha VMS V7.3-2
%HTTPD-I-GMT, +10:30
%HTTPD-I-INSTANCE, supervisor
%HTTPD-I-GBLSEC, created global section of 16 page(let)s
%HTTPD-I-INSTANCE, 1 process
%HTTPD-I-INSTANCE, process name HTTPd:7080
%HTTPD-W-AUTH, 1 informational, 1 warning, 0 errors at load
1.w PROMISCUOUS authenticating any username with specified password!
2.i Cache for 32 records of 768 bytes in local storage of 49 page(let)s
%HTTPD-I-SCRIPTING, as HTTP$NOBODY
%HTTPD-I-DCL, subprocess scripting
%HTTPD-I-ACTIVITY, created global section of 816 page(let)s
%HTTPD-I-SERVICE, http://klaatu.local.org:7080
%HTTPD-I-DEMO, demonstration mode
1.i subprocess scripting
2.i promiscuous authentication
3.i directory access control files ignored
4.i [DirAccess] enabled
5.i [DirMetaInfo] enabled
6.i [DirWildcard] enabled
7.i [Logging] disabled
8.i [ReportBasicOnly] disabled
9.i [ReportMetaInfo] enabled
%HTTPD-I-BEGIN, 02-JAN-2004 21:53:10, accepting requests
```

When *http://the.host.name:7080* is accessed the browser should display something resembling

```
    == / \           Welcome. This page provides some basic
   /W A S D\         information on, and access to, the
Empowered by VMS     "WASD VMS Hypertext Services"
   \/---\__/         package version 8.5
        ==
```

Note

The WASD server which is started by the [INSTALL]DEMO.COM procedure does not have the full environment setup at that time. It is deliberately limited to the single process context. For instance, do not try to execute the command-line directives described in this document.

4.8 “Clone” Procedure

The [INSTALL]CLONE.COM procedure assists in creating a ZIP archive of an existing WASD installation suitable for recreating the server on another system without the necessity of a full installation. This could be used to populate a series of systems with pre-configured servers.

4.9 Re-Linking

After a major update to the operating system the package may refuse to start, reporting a message like:

```
%DCL-W-ACTIMAGE, error activating image WHAT$EVER
-CLI-E-IMGNAME, image file DKA0:[SYS0.SYSCOMMON.][SYSLIB]WHAT$EVER_SHR.EXE
-SYSTEM-F-SHRIDMISMAT, ident mismatch with shareable image
```

This implies the executables require re-linking for your particular version of VMS. This can be accomplished quite simply, perform the linking section only of the update DCL procedure, Section 4.6.

4.10 VMS 6.0 and 6.1

Persona scripting requires a minimum VMS V6.2 to provide the $PERSONA services required for this functionality. If unsure about *persona* scripting please consult the ‘‘Scripting Overview’’ document. Equivalent functionality on earlier versions of VAX VMS is available using the PERSONA_MACRO build option. This will be prompted for by the INSTALL.COM and UPDATE.COM procedures if VAX VMS V6.0 or V6.1 is detected. It is completely optional functionality, the default for these versions is merely to report that persona scripting is unavailable.

A kernel-mode MACRO module is used to provide sufficient functionality to support non-server account scripting. This module makes a momentary modification to the server process username in kernel data structures allowing a detached (scripting) process to be created under that account. The standard WASD server STARTUP.COM procedure will detect whether the MACRO module has been compiled into the executable and INSTALL the image with CMKRNL privilege if required.

Note

Although this approach has been used by a number of tools and applications and has proved quite reliable it is still a mechanism unsupported by the operating system proper and so may have a (potentially) undesirable impact on system integrity.

An alternative is to run the server as a NETWORK mode process.

4.11 VMS 5.5-*n*

WASD is only officially supported for VMS V6.0 or greater. However pre-7.*n* versions have been known to successfully build and run under VMS V5.5-*n*. It will, in all probability, require the AACRTL060 kit (which is part of DECC for this version of VMS, or can be obtained and installed separately).

One issue was a difficulty in using the CGI-BIN logical. This was isolated to the hyphen it contains and resolved by changing the definition of this in STARTUP.COM, using instead ''CGI-BIN''. This is now the default for the example startup procedure, allowing both 5.5-*n* and later VMS versions to function correctly.

4.12 Local Setup Suggestions

Package updates **will never** contain anything in these directories:

HT_ROOT:[HTTP$NOBODY]
HT_ROOT:[HTTP$SERVER]
HT_ROOT:[LOCAL]
HT_ROOT:[STARTUP]

To prevent the overwriting of local configuration files it is suggested these be placed in the HT_ROOT:[LOCAL] directory. Local authentication databases could also be placed in the [LOCAL] directory. Startup files can be placed where-ever the local site manages system startup. These could be placed in the HT_ROOT:[STARTUP] directory.

4.13 Reporting Problems

This package, as is generally the case with freeware, is mainly developed and supported outside of the author's main occupation and working hours. Reports of problems and bugs (while not necessarily welcome :^), as well as general queries, are responded to as soon as practicable. If the documentation is inaccurate or could benefit from clarification in some area please advise of this also (the better the documentation the less queries you have to field personally . . . or so the theory goes).

With all reports please include the version of the server or script, and the hardware platform, operating system and TCP/IP package and version in use.

If a server error message is being generated please examine the HTML source of the error page. The ''<META...>'' information contains version information as well as valuable source code module and line information. Include this with the report.

If the server is exiting with a server-generated error message this information also contains module and line information. Please include this with the report.

The WATCH facility (Chapter 19) is often a powerful tool for problem investigation. It is also very useful when supplying details during problem resolution. **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.

Image crash dumps may also be generated, although these are of less value than the case of the previous two.

Reports may be e-mailed to
Mark.Daniel@wasd.vsm.com.au.au

Should the above address present problems or provide no response for an extended period then use
Mark.Daniel@dsto.defence.gov.au

Chapter 5
Server Account and Environment

The HTTP server account should be a standard account, preferably in a group of its own (definitely at least a non-system, non-user group), with sufficient quotas to handle the expected traffic.

Process Quotas!

Server process quotas must be sufficient to support the expected traffic load. In particlular PRCLM must support expected script usage. BYTLM, BIOLM, DIOL, FILLM and PGFLQUO are all significant considerations.

Symptoms of insufficient process quotas include:

- Textual pages OK, but pages with a significant number of images having some or all ''broken''.
- Scripts failing mysteriously, particularly when multiple in use concurrently.
- Server and associated scripts all apparently waiting MWAIT or RWAST states.

A general rule is more is better, after all, it will only use as much as it needs! To assist with setting a reasonable BYTLM quota the WATCH report provides some feedback on server BYTLM usage. See Chapter 19 for further details.

TCP/IP Agent Resources!

On an associated topic; some TCP/IP agents require particular internal resources to be adjusted against given loads (e.g. buffer space allocations). Symptoms of resource starvation may be TCP/IP services, including WASD, ''pausing'' for significant periods or associated processes entering miscellaneous wait states, etc., during processing. Please ensure such TCP/IP agents are appropriately dimensioned for expected loads.

5.1 VMS Server Account

The following provides a guide to the account.

```
Username: HTTP$SERVER                      Owner:  WASD Server
Account:  HTTPD                            UIC:    [077,001] ([HTTP$SERVER])
CLI:      DCL                              Tables: DCLTABLES
Default:  HT_ROOT:[HTTP$SERVER]
LGICMD:   LOGIN
Flags:  Restricted DisNewMail
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
Primary   000000000011111111112222  Secondary 000000000011111111112222
Day Hours 012345678901234567890123  Day Hours 012345678901234567890123
Network:  ##### Full access ######            ##### Full access ######
Batch:    ##### Full access ######            ##### Full access ######
Local:    -----  No access  ------            -----  No access  ------
Dialup:   -----  No access  ------            -----  No access  ------
Remote:   -----  No access  ------            -----  No access  ------
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:      (pre-expired)
Last Login:            (none) (interactive), 11-MAY-1995 08:44 (non-interactive)
Maxjobs:         0  Fillm:       300  Bytlm:       500000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:      2048  JTquota:       1024
Prclm:         100  DIOlm:      1024  WSdef:         1000
Prio:            4  ASTlm:      2000  WSquo:         5000
Queprio:         0  TQElm:       100  WSextent:     20000
CPU:        (none)  Enqlm:       256  Pgflquo:     500000
Authorized Privileges:
  NETMBX    TMPMBX
Default Privileges:
  NETMBX    TMPMBX
```

5.2 VMS Scripting Account

The following provides a guide to the account.

```
Username: HTTP$NOBODY                       Owner:  WASD Scripting
Account:  HTTPD                             UIC:    [076,001] ([HTTP$NOBODY])
CLI:      DCL                               Tables: DCLTABLES
Default:  HT_ROOT:[HTTP$NOBODY]
LGICMD:   LOGIN
Flags:  Restricted DisNewMail
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
Primary   000000000011111111112222  Secondary 000000000011111111112222
Day Hours 012345678901234567890123  Day Hours 012345678901234567890123
Network:  ##### Full access ######            ##### Full access ######
Batch:    ##### Full access ######            ##### Full access ######
Local:    -----  No access  ------            -----  No access  ------
Dialup:   -----  No access  ------            -----  No access  ------
Remote:   -----  No access  ------            -----  No access  ------
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:       (pre-expired)
Last Login:            (none) (interactive), 11-MAY-1995 08:44 (non-interactive)
Maxjobs:         0  Fillm:       300  Bytlm:       500000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:      2048  JTquota:       1024
Prclm:         100  DIOlm:      1024  WSdef:         1000
Prio:            4  ASTlm:      2000  WSquo:         5000
Queprio:         0  TQElm:       100  WSextent:     20000
CPU:        (none)  Enqlm:       256  Pgflquo:     500000
Authorized Privileges:
  NETMBX    TMPMBX
Default Privileges:
  NETMBX    TMPMBX
```

5.3 Account Support Files

Note

Support procedures often change between versions. It is always advisable to check the versions documentation before installing or updating. Examples may be found in HT_ROOT:[EXAMPLE].

online hypertext link

HTTPd Executables

Two server executables can be built by the package.

- **HTTPD.EXE -** basic server
- **HTTPD_SSL.EXE -** SSL-enabled server (Chapter 17).

Privileged Image

As this image is to be installed with privileges unauthorized use should be prevented by applying an ACL similar to the following against the executable image:

```
$ SET SECURITY HT_EXE:HTTPD.EXE -
  /ACL=((IDENT=HTTP$SERVER,ACCESS=R+E),(IDENT=*,ACCESS=NONE))
```

This can be done once, at installation, or for peace-of-mind *(a.k.a. VMS-ish paranoia)* at each server startup.

As the HTTP$SERVER account should be completely unprivileged, and the HTTPd image requires CMKRNL, NETMBX, TMPMBX, PRMGBL, PRMMBX, PSWAPM, SHMEM (VAX only), SYSGBL, SYSLCK, SYSNAM, SYSPRV and WORLD privileges (see the ‘‘Nuts and Bolts’’ document for a description of how and why the server uses these privileges). It must be installed using a command similar to the following:

```
$ INSTALL = "$SYS$SYSTEM:INSTALL/COMMAND_MODE"
$ INSTALL ADD HT_EXE:HTTPD.EXE -
  /PRIVILEGE=(ALTPRI,CMKRNL, PRMGBL,PRMMBX,PSWAPM,SHMEM,-
              SYSGBL,SYSLCK,SYSNAM,SYSPRV,WORLD)
```

STARTUP.COM

Putting all this together the HTTP server startup procedure becomes something similar to the supplied example. It should be called from SYSTARTUP_VMS.COM or the site’s equivalent.

This procedure will support simple and quite complex sites. It works closely with STARTUP_ SERVER.COM (see below). It is designed to accept parameters from the command-line or as pre-assigned symbols. Operating in this fashion should mean that no modifications will need to be made to the procedure itself. Startup characteristics are essentially determined by DCL symbol values. Some symbols are booleans, switching functionality off and on, others require string values. When relevant startup values are not assigned a reasonable default will be applied. See the following examples.

Startup characteristics can be determined by supplying symbol assignment values as command-line parameters when calling the procedure.

```
$ @$1$DKA0:[HT_ROOT.LOCAL]STARTUP WASD_DECNET=1 WASD_SSL=1 -
WASD_SSL_CERTIFICATE="HT_ROOT:[LOCAL]ALPHA.PEM"
```

Startup characteristics can also be determined by assigning the symbol values before calling the procedure itself.

```
$ WASD_DECNET = 1
$ WASD_SSL = 1
$ WASD_SSL_CERTIFICATE = "HT_ROOT:[LOCAL]ALPHA.PEM"
$ @$1$DKA0:[HT_ROOT.LOCAL]STARTUP
```

On version of VAX VMS prior to 6.2 the startup uses a system batch queue. By default SYS$BATCH is used. If a node does not have a SYS$BATCH then one must be created. If a clustered node’s SYS$BATCH is configured to run on a cluster-common batch queue (i.e. not necessarily on the startup node) then a node-specific queue must be specified.

```
$ @$1$DKA0:[HT_ROOT.LOCAL]STARTUP WASD_DECNET=1 WASD_BATCH_QUEUE=THIS$BATCH
```

Check the procedure itself for detail on symbol names and functionality.

See HT_ROOT:[EXAMPLE]STARTUP.COM

STARTUP_LOCAL.COM

This file is automatically executed by the STARTUP.COM procedure immediately before the server is actually started. It is provided to supply all the local site's additional startup requirements. Place site-specific server environment startup in here, leaving STARTUP.COM alone as much as possible.

See HT_ROOT:[EXAMPLE]STARTUP_LOCAL.COM

STARTUP_SERVER.COM

This procedure serves two purposes.

1. Server startup:
   - If on VAX VMS V6.2 or earlier it is submitted to the SYS$BATCH queue during startup. The batch portion creates a detached process, which then again uses this procedure as input, supporting the executing HTTPd.
   - With more modern versions and architectures of VMS the procedure becomes SYS$COMMAND for a detached process created directly during the execution of STARTUP.COM.
2. The procedure then controls the activation of the HTTPd executable image during normal restarts and exits, and exits after fatal server error.

See HT_ROOT:[EXAMPLE]STARTUP_SERVER.COM

5.4 Other Resources

Other resources required or consumed by the package.

Global Pages/Sections

Accounting and request data made available to the server monitor utility (HTTPDMON) is provided by shared global memory. This requires one global section (SYSGEN parameter GBLSECTIONS) and 16 global pages (SYSGEN parameter GBLPAGES). The activity statistics available from the Server Admininistration facility requires one global section and 816 global pages. These two global sections are permanent.

If multiple server *instances* are to be employed one more global section is required for a standard server (a shared authentication cache), or two more for an SSL server (a shared session cache), with another if reverse proxy verify is enabled, plus a variable number (some tens) of global pages. These global sections are temporary.

If there are insufficient global sections or pages the server will fail to start for all requirements except the activity statistics, this will just be disabled. Startup messages advise on current usage.

As permanent, system-accessable global sections are deployed it may be necessary to explicitly delete them after ad hoc server experimentation, etc. (Section 5.5.1). The startup qualifier /GBLSEC=NOPERM disables the creation of permanent global sections eliminating this requirement.

Logical Names

The following logical names are used in the operation of the HTTPd server and most must be defined before startup (system-wide, or in the job table if server-specific). These are usually created by STARTUP.COM during server startup.

Package Logical Names

Logical Name	Description
HTTPD$AUTH	Location of the authentication/authorization configuration file.
HTTPD$CONFIG	Location of the configuration file.
HTTPD$MAP	Location of the mapping rule file.
HTTPD$MSG	Location of the message file.
HTTPD$SERVICE	Location of the optional service (virtual host) configuration file.
HTTPD$SITELOG	Location of the optional plain-text site log file (Section 18.5).
HTTPD$GMT	Offset from GMT (e.g. ''+10:30'', ''-01:15''). For systems supporting DTSS (e.g. DECnet-Plus) this logical may be left undefined, with server time being calculated using the SYS$TIMEZONE_ DIFFERENTIAL logical.
HTTPD$LOG	If logging is enabled and no log file name specified on the command line, this logical must be defined to locate the file. When a logging period is in use this logical need only contain the directory used to store the logs.
HTTPD$SSL_CERT	When using the SSL executable this logical locates the default certificate.
HTTPD$STARTUP_SERVER	Can be used to pass parameters to HTTPd startup.
AXP_BIN	Directory containing Alpha script executables.
CGI-BIN	System logical defining a search list with the architecture-specific executable directory first, local script directory second, then the common script directory, as a concealed device.
CGI_BIN	Directory containing archtecture-neutral script files.
CGI_EXE	Directory containing architecture-specific script executables.
HT_AXP	Directory containing Alpha executable images.
HT_AUTH	Directory containing authentication/authorization databases (files).
HT_EXE	Directory containing the executable images.
HT_IA64	Directory containing IA64 executable images.
HT_LOGS	Optional definition, for convenient log file specification.

Logical Name	Description
HT_SCRATCH	Location of an optional directory that scripts can use for temporary storage. Must be read+write+delete accessable to the server account. The HTTPD$CONFIG [DclCleanupScratchMinutesMax] directive controls whether automatic cleanup scans of this area delete any files that are older than [DclCleanupScratchMinutesOld].
HT_SERVER_LOGS	Optional definition, for convenient detached server process log file specification.
HT_VAX	Directory containing VAX executable images.
IA64_BIN	Directory containing IA64 script executables.
VAX_BIN	Directory containing VAX script executables.

5.5 HTTPd Command Line

Command-line qualifiers provide some server startup control as well as server runtime control.

5.5.1 Server Startup

When starting up the server several characteristics of the server may be specified using qualifiers on the command line. If not specified appropriate defaults are employed. For recommended methods of passing parameters to the executable at server startup see STARTUP_SERVER.COM.

Server Image Command-Line Parameters

Parameter/Qualifier	Description
/ACCEPT=	Comma-separated list of hosts/domains allowed to connect to the server.
/ALL[=string]	Has two roles. When starting a server up assigns that server to a specific, non-default group of servers (for cluster-wide server control and proxy cache management). When using the server control /DO= using /ALL specifies to do the action to all servers in the group.
/AUTHORIZATION=[SSL,ALL]	The ''SSL'' keyword causes all authentication (both SYSUAF and HTA database) to be available only via ''https:'' requests (Chapter 17). The ''ALL'' keyword forces the server to deny access to any path that does not have authorization in place against it (Section 15.2).

Parameter/Qualifier	Description
/CGI_PREFIX=	The prefix to the CGI symbol names created for a script (defaults to ''WWW_'', similar to the CERN VMS HTTPd, see ''Scripting Environment'' document).
/CLUSTER	Apply control /DO= to all instances in the cluster (default is to per-node instances only).
/DEMO	Places the server into demonstration mode designed to allow full package capabilities to be demonstrated. Used by the [INSTALL]DEMO.COM procedure.
/DETACH=	For VMS 6.2 and later this qualifier allows a DCL procedure to be specified as input to a directly detached process (in conjunction with /USER).
/DO=	Command to be performed by the executing server (Section 5.5.2).
/FILBUF=	Number of bytes in the read buffer for files open for processing (i.e. menu files, image mapping configuration files, pre-processed HTML files, etc., not direct file transfers).
/FORMAT=	Overrides the configuration parameter [LogFormat].
/GBLSEC=DELETE	Allows a monitor-associated permanent global section to be explicitly deleted. When a server starts it creates system-accessable, permanent global sections in which to store accounting and request data. As this is permanent it would be possible for a site, perhaps experimenting with servers over a range of ports, to consume significant amounts of global pages and sections. This qualifier allows such sections to be deleted. See also the /GBLSEC=NOPERM described immediately below.
/GBLSEC=NOPERM	Disables the creation of permanent global sections. They are automatically deleted when the server image exits.
/REJECT=	Comma-separated list of hosts/domains not allowed to connect to the server.
/[NO]LOG[=name]	Either disables logging (overrides configuration directive), or enables logging and optionally specifies the log file name (also see section Logical Names, logging is disabled by default). If the file specification is ''SYS$OUTPUT'' the server issues log entries to <stdout>, allowing user-defined log formats to be easily checked and refined.
/NOMONITOR	Allows the update of the data read by HTTPDMON to be disabled.
/NETBUF=	Minimum number of bytes in the network read buffer.
/NETWORK	Run the server and any scripting processes as NETWORK mode rather than the default detached OTHER mode.

Parameter/Qualifier	Description
/OUTBUF=	Number of bytes in the output buffer (for direct file transfers, buffered output from menu interpretation, HTML-preprocessing, etc.)
/PERIOD=	Overrides the configuration parameter [LogPeriod].
/PERSONA[=*ident-name*, RELAXED,AUTHORIZED, RELAXED=AUTHORIZED]	Enables detached process scripting. When used without the *ident-name* all non-privileged accounts (appropriately mapped of course) may have scripts executed under them. If the optional *ident-name* is supplied it specifies the name of a rights identifier the account must be granted before scripts can be activated under it. The RELAXED, AUTHORIZED and RELAXED=AUTHORIZED further control the use of persona functionality with privileged accounts. See ''Scripting Overview, Introduction'' for further detail.
/PORT=	Overrides the configuration parameter [Port] BUT is in turn overridden by the [Service] configuration parameter and /SERVICE= qualifier (is really only useful for use with the /DO= qualifier).
/PRIORITY=	Server process priority (default is 4).
/[NO]PROFILE	Allows SYSUAF-authenticated username security profiles to be used for file access (Section 15.10).
/PROMISCUOUS[=password]	Server will accept any authentication username/password pair (used for testing, demonstrations, etc.)
/PROXY=string	Allows proxy maintainance activitied to be executed from the command line (e.g. from batch jobs, etc.). See Section 16.2.3.
/SCRIPT=AS=	Specifies the username of the default scripting account.
/SERVICE=	Comma-separated, list of server services (overrides the [Service] configuration parameter).
/[NO]SSL[=version]	Controls Secure Sockets Layer protocol behaviour. The version can be any of ''2'', ''3'' or ''23'' (i.e. both 2 and 3, and the default) specifying which SSL protocol version the server will service.
/SUBBUF=	Number bytes in a (sub)process' SYS$OUTPUT buffer.
/[NO]SWAP	Controls whether the server process may be swapped out of the balance set (default is swapping disabled).

Parameter/Qualifier	Description
/[NO]SYSUAF[=ID, PROXY,SSL,WASD]	Allows or disallows (D) username authentication using the server system's SYSUAF (Section 15.10), the optional ''SSL'' keyword causes SYSUAF authentication to be available only via ''https:'' requests (Chapter 17), the optional ''PROXY'' keyword allows SYSUAF proxying, and the optional ''ID'' keyword makes SYSUAF authentication only available to account possessing a specific identifier (Section 15.10). The ''WASD'' keyword makes the deprecated, "hard-wired" WASD identifier environment available to this server. See Section 15.10.3.
/USER	For VMS 6.2 and later this qualifier allows the /DETACH qualifier to directly create a detached process executing as the specified username.
/VERSION	Displays the executable's version string and the copyright notice.
/[NO]WATCH=	Controls the use of the WATCH reporting facility. See Chapter 19 for further details.

5.5.2 Server Command Line Control

A foreign command for the HTTPD control functionality will need to be assigned in the adminstration users' LOGIN.COM, for example:

```
$ HTTPD == "$HT_EXE:HTTPD"
$ HTTPD == "$HT_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.

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. The same functionality is available from the online Server Administration facility.

5.5.2.1 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
```

5.5.2.2 Authentication

See Chapter 15.

The authorization rule file (HTTP$AUTH) may be reloaded.

```
$ 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 15.9).

```
$ HTTPD /DO=AUTH=PURGE
```

A ''skeleton-key'' username and password may be entered, amongst things allowing access to the Server Administration facility (Chapter 18).

```
$ HTTPD /DO=AUTH=SKELKEY=_<username>:<password>[:<period>]
```

5.5.2.3 Cache

Server cache control may also be exercised from the Server Administration page (Chapter 18). The file cache (Chapter 20) may be enabled, disabled and have it's contents purged (declared invalid and reloaded) using

```
$ HTTPD /DO=CACHE=ON
$ HTTPD /DO=CACHE=OFF
$ HTTPD /DO=CACHE=PURGE
```

5.5.2.4 DCL/Scripting Subprocesses

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 subprocesses, 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 subprocesses and mark busy ones for deletion when completed processing.

```
$ HTTPD /DO=DCL=PURGE
```

5.5.2.5 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
```

5.5.2.6 Instances

The number of server instances (Section 6.2) 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 HTTPD$CONFIG configuration directive [InstanceMax] to once again determine the number of instances required. The same functionality is available from the Server Administration page (Section 18.6).

5.5.2.7 Logging

Server logging control may also be exercised from the server administration menu (Chapter 18).

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
```

OBSOLETE

The following directives have been rendered obsolete due to the increasing complexity of WASD access logging.

```
$ HTTPD /DO=LOG=FORMAT=string
$ HTTPD /DO=LOG=OPEN=file-name
$ HTTPD /DO=LOG=PERIOD=string
$ HTTPD /DO=LOG=REOPEN=file-name
```

5.5.2.8 Mapping

See Chapter 13.

The mapping rule file (HTTPD$MAP) may be reloaded.

```
$ HTTPD /DO=MAP
```

5.5.2.9 Shutdown and Restart

Server shutdown may also be exercised from the Server Administration page (Chapter 18).

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
```

5.5.2.10 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 its resulting prompt. When entered the password is not echoed.

```
$ HTTPD /DO=SSL=KEY=PASSWORD
Enter private key password []:
```

5.5.2.11 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
```

Chapter 6
Server Configuration

Server configuration basically concerns itself with the very fundamental behaviour of the server process. Requirements such as buffer and cache sizes, timeout values, scripting limits, content-types, icons, number and type of services offerted, etc., are determined from configuration information.

By default, the system-table logical name HTTPD$CONFIG locates a common configuration file, unless an individual configuration file is specified using a job-table logical name. Simple editing of this file changes the configuration. Comment lines may be included by prefixing them with the hash ‘‘#’’ character. Configuration file directives are not case-sensitive. Any changes to the configuration file can only be enabled by restarting the HTTPd process using the following command on the server system.

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

A server’s currently loaded configuration can be interrogated. See Chapter 18 for further information.

6.1 Site Organisation

Here are a few ‘‘Mother’s Truths’’ about site organisation. These are only basic and obvious suggestions (after a little step back from the sometimes initially overwhelming feeling of ‘‘what do I do now with this brand new toy?’’). There are lots of general documents of Web site organisation and design that are applicable to all server environments. Above all, bring your own software system design experience to the Web-specific environment, it’s not all that different to any other transaction-based, user-interactive environment.

It is recommended that the server distribution tree and any document and other Web-specific data areas be kept separate and distinct. The former in HT_ROOT:[000000], the latter perhaps in something like WEB:[000000]. This logical device could be provided with the following DCL introduced into the site or server startup procedures:

```
$ DEFINE /SYSTEM /TRANSLATION=CONCEALED WEB DSA811:[WEB.]
```

Note that logical device names like this need not appear in in the structure of the Web site. The root of the Web-accessable path can be concealed using a final mapping rule similar to the following

```
pass /* /web/*
```

which simply defaults *anything else* to that physical area. Of course if that *anything else* needs to exist then it must be located in that physical area.

Mapping rules are the tools used to build a logical structure to a site from the physical area, perhaps multiple areas, used to house the associated files. The logical organisation of served data is largely hierarchical, organised under the Web-server path root, and is achieved via two mechanisms.

1. The natural tree structure provided by a hierarchical file system.
2. The logical hierarchy possible using rules within the mapping file to place disparate physical areas into a single logical structure.

Physically distinct areas are used for good physical reasons (e.g. the area can best be hosted on a task-local disk), for historical reasons (e.g. the area existed before any Web environment existed) or for reasons of convenience (e.g. lets put this where access controls already allow the maintainers to manage it).

Guidelines

There are no good reasons for having site-specific documents integrated with the package directory structure. All site-served files should be located in an autonomous, dedicated area or areas. The only reason to place script files into HT_ROOT:[CGI-BIN] or HT_ROOT:[*architecture*_BIN] is that the script script is traditionally accessable via a /cgi-bin/ path or that the site is a small and/or low usage environment where this directory is conveniently available for the few extra scripts being made available.

For any significant site (size that as best suits your perception), or for when a specific software system or systems is being built or exists and it is being ''Web-ified'', design that software system as you would be any other. That is place the documentation in one directory are, executables and support procedures in their own, management files in another, data in yet another area, etc. Then make those portions that are required to be accessable via the Web interface accessable via the logical associations afforded through the use of the server's mapping rules (Chapter 13). Of course existing areas that are to be now made available via the Web can be mapped in the same way. This includes the active components - executable scripts. There is no reason (apart from historical) why the /cgi-bin/ path should be used to activate scripts associated with a dedicated software system. Use a specific and unique path for scripts associated with each such system.

When making a directory structure available via the Web care must be taken that only the portions required to be accessed can be. Other areas should or must not be accessable. The server process can only access files that are world-accessable, it is specifically granted access via VMS protection mechanisms (e.g. ACLs), or that the individual SYSUAF-authorized accessor can access and which have specifically been made available via server authorization rules. Use the recommendations in Section 7.1 as guidlines when designing your own site's protections and permissions.

Document Root

A particular area of the file system may be specified as the *root* of a particular (virtual) sites documents. This is done using the HTTPD$MAP SET *map=root=<string>* mapping rule. After this rule is applied all subsequent rules have the specified string prefixed to mapped strings before file-system resolution.

For example, the following HTTPD$MAP rule set

```
[[the.virtual.site:*]]

pass /*/-/* /ht_root/runtime/*/*
/ht_root/* /ht_root/*

set * map=root=/dka0/the_site

exec /cgi-bin/* /cgi-bin/*
pass /* /*
fail *
```

when applied to the following request URLs results in the described mappings being applied.

```
http://the.virtual.site/doc/example.txt
```

access to the document represented by file

```
DKA0:[THE_SITE.DOC]EXAMPLE.TXT
```

With the request for a directory icon using

```
http://the.virtual.site/-/httpd/file.gif
```

access to the image represented by file

```
HT_ROOT:[RUNTIME.HTTPD]FILE.GIF
```

And a request for a script using

```
http://the.virtual.site/cgi-bin/example.php
```

activation of the script represented by the file

```
DKA0:[THE_SITE.CGI-BIN]EXAMPLE.PHP
```

Care must be taken in getting the sequence of mapping rules correct for access to non-site resources before actually setting the document root which then ties every other resource to that root.

6.2 Server Instances

The term *instance* is used by WASD to describe an autonomous server process. WASD will support multiple servers running on a single system, alone or in combination with multiple servers running across a cluster. 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 restart the server). Ensure you have the requisite version/ECO/patch installed before activating multiple instances on production systems!

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.

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 (Chapter 21) 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.

Configuration

If not explicitly configured only one instance is created. The configuration directive [Instance-Max] allows multiple instances to be specified (Chapter 8). 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* (Chapter 9).

6.3 Virtual Services

The WASD server is capable of concurrently supporting the same host name on different port numbers and a number of different host names (DNS aliased or multi-homed) using the same port number. This capability is generally known as a *virtual server*. There is no design limitation on the number of these services that WASD will concurrently support. Virtual services offer versatile and powerful multi-site capabilities using the one system and server. Service determination is based on the contents of the request's ''Host:'' header field. If none is present it defaults to base service for the interface's IP address and port.

The same mechanism also effectively allows a single instance of the configuration files to support multiple server processes (using the /SERVICE qualifier), either on the one system or across multiple systems, as in a cluster (i.e. supports ''virtual'' and ''real'' servers.) See STARTUP_SERVER.COM for further information on startup support for these configurations.

WASD provides server process run-time parameters via the HTTPD$CONFIG configuration file. These provide settings for logging, scripting, timeouts, file content-type mappings, etc. The HTTPD$MSG file provides configurable system messages.

[Service]

Using the [Service] HTTPD$CONFIG configuration parameter or the /SERVICE qualifier the server creates an HTTP service for each specified. If the host name is omitted it defaults to the local host name. If the port is omitted it defaults to 80. The first port specified in the service list becomes the ‘‘administration’’ port of the server, using the local host name, appearing in administration reports, menus, etc. This port is also that specified when sending control commands via the /DO= qualifier (Section 5.5.2).

This rather contrived example shows a server configured to provide four services over two host names.

```
[Service]
alpha.wasd.dsto.defence.gov.au
alpha.wasd.dsto.defence.gov.au:8080
beta.wasd.dsto.defence.gov.au
beta.wasd.dsto.defence.gov.au:8000
```

Note that both the HTTPD$SERVICE configuration file (see Chapter 9 and the /SERVICE= command-line qualifier (Section 5.5) override this directive.

HTTPD$SERVICE

If the logical name HTTPD$SERVICE is defined (and it is optional) the HTTPD$CONFIG [Service] directive is not used. As mentioned, this configuration file is optional. For simple sites, those containing one or two services, the use of a separate service configuration file is probably not warranted. Once the number begins to grow this file offers an easier-to-manage interface for those services.

See Chapter 9 for further detail.

[[virtual-server]]

The essential profile of a site is established by it’s mapped resources and any authorization controls, the HTTPD$MAP and HTTPD$AUTH configuration files respectively, and these two files support directives that allow configuration rules to be applied to all virtual services (i.e. a default), to a host name (all ports), or to a single specified service (host name and specific port).

To restrict rules to a specified server (virtual or real) add a line containing the server host name, and optionally a port number, between double-square brackets. All following rules will be applied only to that service. If a port number is not present it applies to all ports for that service name, otherwise only to the service using that port. To resume applying rules to all services use a single asterisk instead of a host name. In this way default (all service) and server-specific rules may be interleaved to build a composite environment, server-specific yet with defaults. Note that service-specific and service-common rules may be mixed in any order allowing common rules to be shared. This descriptive example shows a file with one rule per line.

```
# just an example
this rule applies to all services
so does this
and this one
[[alpha.wasd.dsto.defence.gov.au]]
this one however applies only to ALPHA, but to all ports
as indeed does this
[[beta.wasd.dsto.defence.gov.au:8000]]
now we switch to the BETA service, but only port 8000
another one only applying to BETA
and a third
[[*]]
now we have a couple default rules
that again apply to all servers
```

Both the mapping and authorization modules report if rules are provided for services that are not configured for the particular server process (i.e. not in the server's [Service] or /SERVICE parameter list). This provides feedback to the site administrator about any configuration problems that exist, but may also appear if a set of rules are shared between multiple processes on a system or cluster where processes deliver differing services. In this latter case the reports can be considered informational, but should be checked initially and then occasionally for misconfiguration.

Note

There is a difference when specifying virtual services during service creation and when using them to apply mapping, etc. When creating a service the scheme (or protocol, e.g. ''http:'', ''https:'') needs to be specified so the server can apply the correct protocol to connections accepted at that service. Once a service is created however, it becomes defined by the host-name and port supplied when created. Only one scheme (protocol) can be supported on any one host-name/port instance and so it becomes unnecessary to provide it with mapping rules, etc. The server will complain in instances where it is redundant.

Unknown Virtual Server

If a service is not configured for the particular host address and port of a request one of two actions will be taken.

1. If the configuration directive [ServiceNotFoundURL] is set the request will be redirected to the specified URL. This should contain a specific host name, as well as message page. For the default page use:

   ```
   [ServiceNotFoundURL]  //server.host.name/httpd/-/servicenotfound.html
   ```

2. If the above directive is not set the request is mapped using the default rules (e.g. [[*]]). It is possible to specify a rule set containing a default rule for each virtual server. The unmatched request is then handled by a fallback rule, as illustrated in the following.

```
pass /*/-/admin/*
pass /*/-/* /ht_root/runtime/*/*
exec /cgi-bin/* /cgi-bin/*
[[virtual1.host.name]]
/* /web/virtual1/*
/ /web/virtual1/
[[virtual2.host.name]]
/* /web/virtual2/*
/ /web/virtual2/
[[virtual3.host.name]]
/* /web/virtual3/*
/ /web/virtual3/
[[*]]
/* /web/servicenotfound.html
```

This applies to dotted-decimal addresses as well as alpha-numeric. Therefore if there is a requirement to connect via a numeric IP address such a service must have been configured.

Note also that the converse is possible. That is, it's possible to configure a service that the server cannot ever possibly respond to because it does not have an interface using the IP address represented by the service host.

6.4 Request Throttling

Request ''throttling'' is a term adopted to describe controlling the number of requests that can be processing against any specified path at any one time. Requests in excess of this value are First-In-First-Out (FIFO) queued, up to an optional limit, waiting for a currently processing request to conclude allowing the next queued request to resume processing. This is primarily intended to limit concurrent resource-intensive script execution but could be applied to any resource path. Here's one dictionary description.

throttle n 1: *a valve that regulates the supply of fuel to the engine [syn: accelerator, throttle valve]* ***2:*** *a pedal that controls the throttle valve; ''he stepped on the gas'' [syn: accelerator, accelerator pedal, gas pedal, gas, gun]* ***v 1:*** *place limits on; ''restrict the use of this parking lot'' [syn: restrict, restrain, trammel, limit, bound, confine]* ***2:*** *squeeze the throat of; ''he tried to strangle his opponent'' [syn: strangle, strangulate]* ***3:*** *reduce the air supply; of carburetors [syn: choke]*

This is applied to a path (or paths) using the HTTPD$MAP mapping SET THROTTLE= rule (Section 13.4.5). The general format is

```
set path throttle=n1[,n2,n3,n4,t/o1,t/o2]
set path throttle=from[,to,resume,busy,t/o-queue,t/o-busy]
```

where

- *n1* sets the number of concurrent requests before queuing begins (the number of processing requests becomes static and the number of queued requests increases)
- *n2* is the concurrent requests before FIFO queuing begins, meaning each new request is put onto the queue but at the same the first-in request is taken off the queue for processing (the number of queued requests becomes static and the number of processing requests increases)
- *n3* puts a limit on FIFO queuing (the number of queued requests again increases and the number of processing requests becomes static)

- *n4* is an absolute limit for concurrent requests against the path (a 503 ‘‘server too busy’’ status is immediately generated)
- *t/o1* is the maximum period for queued requests before they are processed (if not constrained by *n3*)
- *t/o2* is the maximum period for queued requests before a 503 ‘‘server too busy’’ response is returned, it begins immediately or following the expiry of any *t/o1*

One way to read a throttle rule is ‘‘begin to *throttle* (queue) requests *from* the n1 value up *to* the n2 value, after which the queue is FIFOed up to the n3 value when it *resume*s queuing-only, up until the *busy* n4 value’’.

Each integer represents the number of concurrent requests against the throttle rule path. Parameters not required may be specified as zero or omitted in a comma-separated list. The schema of the rule requires that each successive parameter be larger than that preceding it. This basic consistency check is performed when the rule is loaded.

For any rule the possible maximum number of requests that can be processed at any one time may be simply calculated through the addition of the *n1* value to the difference of the *n3* and *n2* values (i.e. max = n1 + (n3 - n2)). The maximum concurrently queued as the difference of the *n4* and the maximum concurrently processed.

A comprehensive throttle statistics report is available from the Server Administration page (Chapter 18).

Examples

1. **throttle=10**

   Requests up to 10 are concurrently processed. When 10 is reached futher requests are queued to server capacity.

2. **throttle=10,20**

   Concurrent requests to 10 are processed immediately. From 11 to 20 requests are queued. After 20 all requests are queued but also result in a request FIFOing off the queue to be processed (queue length is static, number being processed increases to server capacity).

3. **throttle=15,30,40**

   Concurrent requests up to 15 are immediately processed. Requests 16 through to 30 are queued, while 31 to 40 requests result in the new requests being queued and waiting requests being FIFOed into processing. Concurrent requests from 41 onwards are again queued, in this scenario to server capacity.

4. **throttle=10,20,30,40**

   Concurrent requests up to 10 are immediately processed. Requests 11 through to 20 will be queued. Concurrent requests from 21 to 30 are queued too, but at the same time waiting requests are FIFOed from the queue (resulting in 10 (n1) + 10 (n3-n2) = 20 being processed). From 31 onwards requests are just queued. Up to 40 concurrent requests may be against the path before all new requests are immediately returned with a 503 "busy" status. With this scenario no more than 20 can be concurrently processed with 20 concurrently queued.

5. **throttle=10,,,30**

   Concurrent requests up to 10 are processed. When 10 is reached requests are queued up to request 30. When request 31 arrives it is immediately given a 503 "busy" status.

6. **throttle=10,20,30,40,00:02:00**

   This is basically the same as scenario 4) but with a resume-on-timeout of two minutes. If there are currently 15 (or 22 or 28) requests (n1 exceeded, n3 still within limit) the queued requests will begin processing on timeout. Should there be 32 processing (n3 has reached limit) the request will continue to sit in the queue. The timeout would not be reset.

7. **throttle=15,30,40,,,00:03:00**

   This is basically the same as scenario 3) but with a busy-on-timeout of three minutes. When the timeout expires the request is immediately dequeued with a 503 "busy" status.

Mapping Reload

Throttling is applied using mapping rules. The set of these rules may be changed within an executing server using map reload functionality. This means the number of, and/or contents of, throttle rules may change during server execution. The throttle functionality needs to be independent of the the mapping functionality (requests are processed independently of mapping rules once the rules have been applied). After a mapping reload the contents of the throttle data structures may be at variance with the constraints currently executing requests began processing under.

This should have little deleterious effect. The worst case is mis-applied constraints on the execution limits of changed request paths, and slightly confusing data in the Throttle Report. This quickly passes as requests being processed under the previous throttle constraints conclude and an entirely new collection of requests created using the constraints of the currently loaded rules are processed.

6.5 Client Concurrency

The ‘‘client_connect_gt:’’ mapping conditional (Chapter 12) attempts to allow some measurement of the number of requests a particular client currently has being processed. Using this decision criterion appropriate request mapping for controlling the additional requests can be undertaken. It is not intended to provide fine-grained control over activities, rather just to prevent a single client using an unreasonable proportion of the resources.

For example. If the number of requests from one particulat client looks like it has got out of control (at the client end) then it becomes possible to queue (throttle) or reject further requests. In HTTPD$MAP

```
if (client_connect_gt:15) set * throttle=15
if (client_connect_gt:15) pass * "503 Exceeding your concurrency limit!"
```

While not completely foolproof it does offer some measure of control over gross client concurrency abuse or error.

6.6 Content-Type Configuration

HTTP uses an implementation of the MIME (Multi-purpose Internet Mail Extensions) specification for identifying the type of data returned in a response. A MIME content-type consists of a plain text string describing the data as a *type* and slash-separated *subtype*, as illustrated in the following examples:

```
text/html
text/plain
image/gif
image/jpeg
application/octet-stream
```

The content-type is returned to the client as part of the HTTP response, the client then using this information to correctly process and present the data contained in that response.

6.6.1 Adding Content-Types

In common with most HTTP servers WASD uses a file's suffix (extension, type, e.g. ''.HTML, ''.TXT'', ''.GIF'''' to identify the data type within the file. The [AddType] directive is used during configuration to bind a file type to a MIME content-type. To make the server recognise and return specific content-types these directives map file types to content-types.

With the VMS file system there is no effective file characteristic or algorithm for identifying a file's content without an exhaustive examination of the data contained there-in . . . a very expensive process (and probably still inconclusive in many cases), hence the reliance on the file type.

Note

When adding a totally new content-type to the configuration be sure also to bind an icon to that type using the [AddIcon] directive (see below). If this is not done the default icon specified by [AddDefaultIcon] is displayed. If that is not defined then a directory listing shows ''[?]'' in place of an icon.

Mappings using [AddType] look like these.

```
[AddType]
.html  text/html   HyperText Markup Language
.txt   text/plain  plain text
.gif   image/gif   image (GIF)
.hlb   text/x-script /Conan  VMS Help library
.decw$book   text/x-script   /HyperReader    Bookreader book
*  internal/x-unknown   application/octet-stream
```

6.6.2 MIME.TYPES

To allow the server to share content-type definitions with other MIME-aware applications, and for WASD scripts to be able to perform their own mapping on a shared understanding of MIME content it is possible to move the file suffix to content-type mapping from a collection of [AddType]s in HTTPD$CONFIG to an external file. This file is usually named MIME.TYPES and is specified in HTTPD$CONFIG using the [AddMimeTypesFile] directive.

Mappings using MIME.TYPES look like these.

```
# MIME type   Extension
application/msword            doc
application/octet-stream      bin dms lha lzh exe class
application/oda               oda
application/pdf               pdf
application/postscript        ai eps ps
application/rtf               rtf
```

A leading content-type is mapped to single or multiple file suffixes. A general MIME.TYPES file commonly has content-types listed with no corresponding file suffix. These are ignored by WASD. Where a file suffix is repeated during configuration the latter version completely supercedes the former (with the Server Administration page showing an italicised and struck-through content-type to help identify duplicates).

To allow the configuration information used by the server to generate directory listings with additional detail, WASD-specific extensions to the standard MIME.TYPES format are provided. These are ''hidden'' in comment structures so as not to interfere with non-WASD application use. All begin with a hash then an exclamation character (''#!'') then another reserved character indicating the purpose of the extension. Existing comments are unaffected provided the second character is anything but an exclamation mark!

- **#! file description**
  A space reserved character indicates following free-form text, used as the file type description displayed on the far right of directory listings.
- **#!/cgi-bin/script**
  A forward-slash introduces an auto-script specification. An auto-script is automatically activated by the server to process and display a corresponding file's contents. These are sometimes refered to as *presentation* scripts.
- **#![*alt*] /path/to/icon.gif**
  A left-square-bracket is used for icon specifications. These are actually mapped against the following content-type, not file suffix, and so only need to be specified once for each content-type in the file. This behaves in a similar fashion to [AddIcon], only the components are reversed.
- **#!!**
  The two exclamation marks can be used to indicate a MIME type intended for WASD only. The can be ignored by non-WASD applications.
- **#!+**
  An exclamation mark then a plus symbol indicates an FTP transfer mode directive. One of three characters may follow the plus. An ''A'' indicates that this file type should be FTP transfered in ASCII mode. An ''I'' or a ''B'' indicates that this file type should be FTP transfered in Image (binary) mode.
- **#!%**
  A percentage is ignored by WASD. This is reserved for local (non-WASD) viewers.

These directives are placed **following** the MIME-type entry they apply to. An example of the contents of a MIME.TYPES file with various WASD extensions.

```
# MIME type   Extension
application/msword            doc
#! MS Word document
#![DOC] /httpd/-/doc.gif
application/octet-stream      bin dms lha lzh exe class
#! binary content
#![BIN] /httpd/-/binary.gif
application/oda               oda
application/pdf               pdf
application/postscript        ai eps ps
#! Adobe PostScript
#![PS.] /httpd/-/postscript.gif
#!+A
application/rtf               rtf
#! Rich Text Format
#![RTF] /httpd/-/rtf.gif
application/x-script          bks decw$bookshelf
#! DEC Bookshelf
#!/cgi-bin/hypershelf
application/x-script          bkb decw$book
#![BKR] /httpd/-/script.gif
#! DEC Book
#!/cgi-bin/hyperreader
```

Other reserved characters have been specified for development purposes but are not (perhaps currently) employed by the HTTP server.

- **#!< html marked-up text**
  A less-than symbol indicates HTML marked-up text.
- **#!# blah blah blah**
  ##! rhubarb rhubarb
  Two combinations of hash and exclamation characters provide for WASD-specific comments.

6.6.3 Unknown Content-Types

If a file type is not recognised (i.e. no [AddType] or [AddMimeTypesFile] mapping corresponding to the file type) then by default WASD identifies its data as *application/octet-stream* (i.e. essentially binary data). Most browsers respond to this content-type with a download dialog, allowing the data to be saved as a file. Most commonly these unknown types manifest themselves when authors use ‘‘interesting’’ file names to indicate their purpose. Here are some examples the author has encountered:

```
README.VMS
README.1ST
READ-ME.FIRST
BUILD.INSTRUCTIONS
MANUAL.PT1 (.PT2,  . . .  )
```

If the site administrator would prefer another default content-type, perhaps ‘‘text/plain’’ so that any unidentified files default to plain text, then this may be configured by specifying that content-type as the *description* of the catch-all file type entry. Examples (use one of):

```
[AddType]
*  internal/x-unknown
*  internal/x-unknown  application/octet-stream
*  internal/x-unknown  text/plain
*  internal/x-unknown  something/else-entirely
```

It is the author’s opinion that unidentified file types should remain as binary downloads, not ‘‘text’’ documents, which they are probably more often not, but it’s there if it’s wanted.

6.6.4 Explicitly Specifying Content-Type

When accessing files it is possible to explicitly specify the identifying content-type to be returned to the browser in the HTTP response header. Of course this does not change the actual content of the file, just the header content-type! This is primarily provided to allow access to plain-text documents that have obscure, non-‘‘standard’’ or non-configured file extensions.

It could also be used for other purposes, ‘‘forcing’’ the browser to accept a particular file as a particular content-type. This can be useful if the extension is not configured (as mentioned above) or in the case where the file contains data of a known content-type but with an extension conflicting with an already configured extension specifying data of a different content-type.

Enter the file path into the browser’s URL specification field ("Location:", "Address:"). Then, for plain-text, append the following query string:

```
?httpd=content&type=text/plain
```

For another content-type substitute it appropriately. For example, to retrieve a text file in binary (why I can’t imagine :^) use

```
?httpd=content&type=application/octet-stream
```

This is an example:

online demonstration

It is posssible to "force" the content-type for all files in a particular directory. Enter the path to the directory and then add

```
?httpd=index&type=text/plain
```

(or what-ever type is desired). Links to files in the listing will contain the appropriate ‘‘?httpd=content&type=...’’ appended as a query string.

This is an example:

6.7 Language Variants

Language-specific variants of a document may be configured to be served automatically and transparently. This is organized as a basic file and name with language-specific variant indicated by an additional ''tag'', one of ISO language abbreviations used by the ''Accept-Language:'' request header field, e.g. *en* for English, *fr* for French, *de* for German, *ru* for Russian, etc.

Two variants of the basic file specification are possible; file name (the default) and file type. Hence if the basic file name is EXAMPLE.HTML then specifically German, English, French and Russian language versions in the directory would be either

```
EXAMPLE.HTML
EXAMPLE_DE.HTML
EXAMPLE_EN.HTML
EXAMPLE_FR.HTML
EXAMPLE_RU.HTML
```

or

```
EXAMPLE.HTML
EXAMPLE.HTML_DE
EXAMPLE.HTML_EN
EXAMPLE.HTML_FR
EXAMPLE.HTML_RU
```

A path must be explicitly SET using the *accept=lang* mapping rule as containing language variants. As searching for variants is a relatively expensive operation the rule(s) applying this functionality should be carefully crafted. The *accept=lang* rule accepts an optional default language representing the contents of the basic, untagged files. This provides an opportunity to more efficiently handle requests with a language first preference matching that of the default. In this case no variant search is undertaken, the basic file is simply served. The following example sets a path to contain files with a default language of French and possibly containing other language variants.

```
set /web/doc/* accept=lang=(default=fr)
```

In this case the behaviour would be as follows. With the default language set to ''fr'' a request's ''Accept-Language:'' field is initially processed to check if the first preference is for ''fr''. If it is then there is no need for further accept language processing and the basic file is returned as the response. If not then the directory is searched for other files matching the EXAMPLE_ *.HTML specification. All files matching this wildcard have the ''*'' portion (e.g. ''EN'', ''FR'', ''DE'', ''RU'') added to a list of variants. When the search is complete this list is compared to the request's ''Accept-Language:'' list. The first one to be matched has the contents of the corresponding file returned. If none are matched the default version would be returned.

This example of the behaviour is based on the contents of the directory described above. A request that specifies

```
Accept-Language: fr,de,en
```

will have EXAMPLE.HTML returned (without having searched for any other variants). For a request specifying

```
Accept-Language: ru,en
```

then the EXAMPLE_RU.HTML file is returned, and if no ''Accept-Language:'' is supplied with the request EXAMPLE.HTML would be returned. One or other file is always returned, with the default, non-language file always the fallback source of data. If it does not exist and no other language variant is selected the request returns a 404 file-not-found error.

Content-Type

When using the *accept=lang=(variant=type)* form of the rule (i.e. the variant is placed on the file type rather than the default file name) each possible file extension must also must have it's content-type made known to the server. Using the example above the variants would need to be configured in a similar way to the following.

```
[AddType]
.HTML     "text/html; charset=ISO-8859-1"  HyperText Markup Language
.HTML_DE  "text/html; charset=ISO-8859-1"  HTML (German)
.HTML_EN  "text/html; charset=ISO-8859-1"  HTML (English)
.HTML_FR  "text/html; charset=ISO-8859-1"  HTML (French)
.HTML_RU  "text/html; charset=koi8-r"      HTML (Russian)
```

Non-Text Content

Normally only files with a content-type of ''text/..'' are subject to variant searching. If the rule path includes a file type then those files matching the rule are also variant-searched. In this way images, audio files, etc., may also have language-specific versions supplied transparently. The following illustrates this usage

```
set /web/doc/*.jpg accept=lang=(default=fr)
set /web/doc/*.wav accept=lang=(default=fr)
```

6.8 Character Set Conversion

The default character set sent in the response header for text documents (plain and HTML) is set using the [CharsetDefault] directive and/or the SET charset mapping rule. English language sites should specify ISO-8859-1, other Latin alphabet sites, ISO-8859-2, 3, etc. Cyrillic sites might wish to specify ISO-8859-5 or KOI8-R, and so on.

Document and CGI script output may be dynamically converted from one character set to another using the standard VMS NCS conversion library. The [CharsetConvert] directive provides the server with character set aliases (those that are for all requirements the same) and which NCS conversion function may be used to convert one character set into another.

```
document-charset  accept-charset[,accept-charset..] [NCS-function-name[=factor]]
```

When this directive is configured the server compares each text response's character set (if any) to each of the directive's *document charset* string. If it matches it then compares each of the *accepted charset* (if multiple) to the request ''Accept-Charset:'' list of accepted characters sets.

At least one *doc-charset* and one *accept-charset* must be present. If only these two are present (i.e. no *NCS-conversion-function*) it indicates that the two character sets are aliases (i.e. the same set of characters, different name) and no conversion is necessary.

If an *NCS-conversion-function* is supplied it indicates that the document *doc-charset* can be converted to the request ‘‘Accept-Charset:’’ preference of the *accept-charset* using the NCS conversion function name specified.

A *factor* parameter can be appended to the conversion function. Some conversion functions require more than one output byte to represent one input byte for some characters. The ’factor’ is an integer between 1 and 4 indicating how much more buffer space may be required for the converted string. It works by allocating that many times more output buffer space than is occupied by the input buffer. If not specified it defaults to 1, or an output buffer the same size as the input buffer.

Multiple comma-separated *accept-charset*s may be included as the second component for either of the above behaviours, with each being matched individually. Wildcard ‘‘*’’ and ‘‘*%*’’ may be used in the *doc-charset* and *accept-charset* strings.

```
[CharsetConvert]
windows-1251 windows-1251,cp-1251
windows-1251 koi8-r windows1251_to_koi8r
koi8-r koi8-r,koi8
koi8-r windows-1251,cp-1251 koi8r_to_windows1251
koi8-r utf-8 koi8r_to_utf8=2
```

6.9 Error Reporting

By default the server provides it’s own internal error reporting facility. These reports may be configured as *basic* or *detailed* on a per-path basis, as well as determining the basic ‘‘look-and-feel’’. For more demanding requirements the [ErrorReportPath] configuration directive allows a redirection path to be specified for error reporting, permitting the site administrator to tailor both the nature and format of the information provided. A Server Side Include document, CGI script or even standard HTML file(s) may be specified. Generally an SSI document would be recommended for it’s simplicity yet versatility.

6.9.1 Basic and Detailed

Internally generated error reports are the most efficient. These can be delivered with two levels of error information. The default is more detailed.

```
ERROR 404 - The requested resource could not be found.
Document not found ... /ht_root/index.html
(document, bookmark, or reference requires revision)
Additional information: 1xx, 2xx, 3xx, 4xx, 5xx, Help
------------------------------------------------------------------
WASD/7.0.0 Server at wasd.dsto.defence.gov.au Port 80
```

There is also the more basic.

```
ERROR 404 - The requested resource could not be found.
Additional information: 1xx, 2xx, 3xx, 4xx, 5xx, Help
------------------------------------------------------------------
WASD/7.0.0 Server at wasd.dsto.defence.gov.au Port 80
```

These can be set per-server using the [ReportBasicOnly] configuration directive, or on a per-path basis in the HTTPD$MAP configuration file. The basic report is intended for environments where traditionally a minimum of information might be provided to the user community, both to reduce site configuration information leakage but also where a general

user population may only need or want the information that a document was either found or not found. The detailed report often provides far more specific information as to the nature of the event and so may be more appropriate to a more technical group of users. Either way it is relatively simple to provide one as the default and the other for specific audiences. Note that the detailed report also includes in page <META> information the code module and line references for reported errors.

To default to a basic report for all but selected resource paths introduce the following to the top of the HTTPD$MAP configuration file.

```
# default is basic reports
set /* report=basic
set /internal-documents/* report=detailed
set /other/path/* report=detailed
```

To provide the converse, default to a detailed report for all but selected paths use the following.

```
# default is detailed reports
set /web/* report=basic
```

Other Customization

The additional reference information included in the report may be disabled using the appropriate HTTPD$MSG [status] message item. Emptying this message results in an error report similar to the following.

```
ERROR 404 - The requested resource could not be found.
-------------------------------------------------------------
WASD/7.0.0 Server at wasd.dsto.defence.gov.au Port 80
```

The server signature may be disabled using the HTTPD$CONFIG [ServerSignature] configuration directive. This results in a minimal error report.

A simple approach to providing a site-specific ''look-and-feel'' to server reports is to customize the [ServerReportBodyTag] HTTPD$CONFIG configuration directive. Using this directive report page background colour, background image, text and link colours, etc., may be specified for all reports. It is also possible to more significantly change the report format and contents (within some constraints), without resorting to the site-specific mechansims refered to below, by changing the contents of the appropriate HTTPD$MSG [status] item. This should be undertaken with care.

```
ERROR 404 - The requested resource could not be found.
```

6.9.2 Site Specific

Customized error reports can be generated for all or selected HTTP status status associated with errors reported by the server using the HTTPD$CONFIG [ErrorReportPath] and HTTPD$SERVER [ServiceErrorReportPath] configuration directives. To explicitly handle all error reports specify the path to the error reporting mechanism (see description below) as in the following example.

```
[ErrorReportPath] /httpd/-/reporterror.shtml
```

To handle only selected error reports add the HTTP status codes following the report path. In this example only 403 and 404 errors are explicitly handled, the rest remain server-generated. This is particularly useful for static error documents.

```
[ErrorReportPath] /httpd/-/reporterror.shtml 403 404
```

Site-specific error reporting works by internal redirection. When an error is reported the original request is concluded and the request reconstructed using the error report path before internally being reprocessed. For SSI and CGI script handlers error information becomes available via a specially-built query string, and from that as CGI variables in the error report context. One implication is the original request path and query string are no longer available. All error information must be obtained from the error information in the new query string.

It is suggested with any use of this facility the reporting document(s) be located somewhere local, probably HT_ROOT:[RUNTIME.HTTPD], and then enabled by placing the appropriate path into the [ErrorReportPath] configuration directive.

```
[ErrorReportPath] /httpd/-/reporterror.shtml
```

Note that virtual services can subsequently have this path mapped to other documents (or even scripts) so that some or all services may have custom error reports. For instance the following arrangement provides each host (service) with an customized error report.

```
# HTTPD$CONFIG
[ErrorReportPath] /errorreport.shtml

# HTTPD$MAP
[[alpha.wasd.dsto.gov.au]]
pass /errorreport.shtml /httpd/-/alphareport.shtml
[[beta.wasd.dsto.gov.au]]
pass /errorreport.shtml /httpd/-/betareport.shtml
[[gamma.wasd.dsto.gov.au]]
pass /errorreport.shtml /httpd/-/gammareport.shtml
```

Using Static HTML Documents

Static HTML documents are a good choice for site-specific error messages. They are very low overhead and are easily customizable. One per possible response error status code is required. When providing an error report path including a ''!UL'' introduces the response status code into the file path, providing a report path that includes a three digit number representing the HTTP status code. A file for each possible or configured code must then be provided, in this example for 403 (authorization failure), 404 (resource not found) and 502 (bad gateway/script).

```
[ErrorReportPath] /httpd/-/reporterror!UL.html 403 404 502
```

This mapping will generate paths such as the following, and require the three specified to respond to those errors.

```
/httpd/-/reporterror403.html
/httpd/-/reporterror404.html
/httpd/-/reporterror502.html
```

Using an SSI Document

SSI documents provide the versatility of dynamic report generation for but they do take time and CPU for processing, and this may be a significant consideration on busy sites.

Three example SSI error report documents are provided. See HT_ROOT:[EXAMPLE]REPORTERROR*.SHTML The first providing a report identical with those internally generated, the second a small variation on this, and the third considerably different and with much less specific error information (which some administrator's may consider advantageous).

The following SSI variables are available specifically for generating error reports. The <!– #printenv –> statement near the top of the file may be uncommented to view all SSI and CGI variables available.

Error Variables

Variable	Description
ERROR_LINE	The HTTPd source code line from where the error was generated.
ERROR_MODULE	The HTTPd source code module corresponding to the line described above.
ERROR_REPORT	A single HTML string providing a detailed error message.
ERROR_REPORT2	A single HTML comment providing more detailed VMS error information if available
ERROR_REPORT3	A server-generated HTML string providing a brief explanation of the error if available
ERROR_STATUS_CLASS	Essentially the single hundreds digit from the status code (e.g. 4).
ERROR_STATUS_CODE	The HTTP response status code representing the error (e.g. 404).
ERROR_STATUS_ EXPLANATION	The HTTP response status code descriptive meaning (e.g. ''The requested resource could not be found.'')
ERROR_STATUS_TEXT	The HTTP response status code abbreviated meaning (e.g. ''Not Found'').
ERROR_STATUS_TYPE	''basic'' or ''detailed''.
FORM_ERROR_ . . .	A series of CGI variables providing the sources for the above SSI variables, as well as other general environment information.

Using a Script

It is also possible to report using a script. The same error information is available via corresponding CGI variables. The source code HT_ROOT:[SRC.MISC]REPORTERROR.C provides such an implementation example.

6.10 OPCOM Logging

Significant server events may be optionally displayed via a selected operator's console and recorded in the operator log. Various categories of these events may be selectively enabled via HTTPD$CONFIG directives (Chapter 8).

- Server Administration page directives
- authentication/authorization (e.g. failures)
- CLI HTTPd control directives
- HTTPd events (e.g. startup, exit, SSL private key password requests)
- proxy file cache maintenance

Some significant server events are always logged to OPCOM if any one of the above categories is enabled.

6.11 Access Logging

WASD provides a versatile access log, allowing data to be collected in Web-standard *common* and *combined* formats, as well as allowing customization of the log record format. It is also possible to specify a log period. If this is done log files are automatically changed according to the period specified.

Where multiple access log files are generated with per-instance, per-period and/or per-service logging (see below) these can be merged into single files for administrative or archival purposes using the CALOGS utility (Section 23.6).

The Quick-and-Dirty LOG STATisticS utility (Section 23.10) can be used to provide elementary ad hoc log analysis from the command-line or CGI interface.

Exclude requests from specified hosts using the [LogExcludeHosts] configuration parameter.

6.11.1 Log Format

The configuration parameter [LogFormat] and the server qualifier /FORMAT specifies one of three pre-defined formats, or a user-definable format. Most log analysis tools can process the three pre-defined formats. There is a small performance impost when using the user-defined format, as the log entry must be specially formatted for each request.

- **COMMON -** This is the most common, base logging format for Web servers. COMMON is the default log format.
- **COMMON_SERVER -** This is an optional format used, for one, by the NCSA server. It is basically the common format, with the server host name appended to the line (used for multi-homed servers, see Section 6.3).
- **COMBINED -** This is an optional format used, for one again, by the NCSA server. It too is basically the common format, with the HTTP referer and user agent appended.

User-Defined

The user-defined format allows customised log formats to be specified using a selection of commonly required data. The specification must begin with a character that is used as a substitute when a particular field is empty (use "\0" for no substitute, as in the "windows log format" example below).

Two different "escape" characters introduce the following parameters:

A ‘‘!’’ followed by

Characters	Description
AR	authentication realm (if any)
AU	authenticated user name (if any)
BB	bytes in body (excludes response header)
BY	bytes in response (includes header)
CA	client address
CN	client host name (or address if DNS lookup disabled)
EM	request elapsed time in milliseconds
ES	request elapsed time in fractional seconds
ID	session track ID
ME	request method
PA	request path (not to be confused with ‘‘RQ’’)
QS	request query string (if any)
RF	referer (if any)
RQ	complete request string (see below)
RS	response status code
SN	server host name
SC	script name (if any)
SM	request scheme (http: or https:)
SP	server port
TC	request time (common log format)
TG	request time (GMT)
TV	request time (VMS format)
UA	user agent

A ‘‘\’’ followed by

Character	Description
0	a null character (used to define the empty field character)
!	insert an ‘‘!’’
\	insert a ‘‘\’’
n	insert a newline
q	insert a quote (so that in DCL the quotes won’t need escaping!)
t	insert a TAB

Any other character is directly inserted into the log entry.

“PA” and “RQ”

The ‘‘PA’’ and ‘‘RQ’’ have distinct roles. In general the ‘‘RQ’’ (request) directive will always be used as this is the full request string; script component (if any), path string and query string component (if any). The ‘‘PA’’ directive is merely the path string after any script and query string components have been removed.

Examples

1. The equivalent of the common log format is:

```
-!CN - !AU [!TC] \q!RQ\q !RS !BY
```

2. The combined log format could be specified as:

```
-!CN - !AU [!TC] \q!RQ\q !RS !BY \q!RF\q \q!UA\q
```

3. The *O’Reilly WebSite* ‘‘windows log format’’ would be created by:

```
\0!TC\t!CA\t!SN\t!AR\t!AU\t!ME\t!PA\t!RQ\t!EM\t!UA\t!RS\t!BB\t
```

4. The common log format with appended request duration in seconds could be provided using:

```
-!CN - !AU [!TC] \q!RQ\q !RS !BY !ES
```

6.11.2 Log Per-Period

The access log file may have a period specified against it, producing an automatic generation of log file based on that period. This allows logs to be systematically named, ordered and kept to a managable size. The period specified can be one of

- DAILY
- weekly as . . .
  MONDAY
  TUESDAY
  WEDNESDAY
  THURSDAY
  FRIDAY
  SATURDAY
  SUNDAY

- MONTHLY

The log file changes on the first request after midnight of the new period. When using a weekly period the new log file comes into effect on the first request following midnight on the specified day.

When using a periodic log file, the file name specified by HTTPD$LOG or the configuration parameter [LogFile] is partially ignored, only partially because the directory component of it is used to located the generated file name. The periodic log file name generated comprises

- server host name
- server port
- year (YYYY)
- month (MM)
- day (DD)

as in the following example

```
HT_LOGS:WASD_80_19971013_ACCESS.LOG
```

For the daily period the date represents the request date. For the weekly period it is the date of the previous (or current) day specified. That is, if the request occurs on the Wednesday for a weekly period specified by Monday the log date show the last Monday's. For the monthly period it uses the first.

6.11.3 Log Per-Service

By default a single access log file is created for each HTTP server process. Using the [LogPerService] configuration directive a log file for each service provided by the HTTPd is generated (Section 6.3). The [LogNaming] format can be any of "NAME" (default) which names the log file using the first period-delimited component of the IP host name, "HOST" which uses as much of the IP host name as can be accomodated within the maximum 39 character filename limitation (of ODS-2), or "ADDRESS" which uses the full IP host address in the name. Both HOST and ADDRESS have hyphens substituted for periods in the string. If these are specified then by default the service port follows the host name component. This may be suppressed using the [LogPerServiceHostOnly] directive, allowing a minimum extra 3 characters in the name, and combining entries for all ports associated with the host name (for example, a standard HTTP service on port 80 and an SSL service on port 443 would have entries in the one file).

6.11.4 Log Per-Instance

To reduce physical disk activity, and thereby significantly improve performance, the RMS characteristics of the logging stream are set to buffer records for as long as possible and only write to disk when buffer space is exhausted (a periodic flush ensures records from times of low activity are written to disk). However when multiple server processes (either in the case of multiple instances on a single node, single instance on each of multiple clustered nodes, or a combination of the two) have the same log files open for write then this buffering and defered write-to-disk is disabled by RMS, it insisting that all records must be flushed to disk for correct serialization and coherency.

This introduces measuraable latency and a potentially significant bottleneck to high-demand processing. Note that it only becomes a real issue under load. Sites with a low load should not experience any impact.

Sites that may be affected by this issue can revert to the original buffered log stream by enabling the [LogPerInstance] configuration directive. This ensures that each log stream has only one writer by creating a unique log file for each instance process executing on the node and/or cluster. It does this by appending the node and process name to the file type. This would change the log name from something like

```
HT_LOGS:131-185-250-202_80_ACCESS.LOG
```

to, in the case of a two-instance single node,

```
HT_LOGS:131-185-250-202_80_ACCESS.LOG_KLAATU_HTTPD-80
HT_LOGS:131-185-250-202_80_ACCESS.LOG_KLAATU_HTTPE-80
```

Of course the number-of and naming-of log files is beginning to become a little itimidating at this stage! To assist with managing this seeming plethora of access log files is the calogs utiltiy (Section 23.6), which allows multiple log files to be merged whilst keeping the records in timestamp order.

6.11.5 Log Naming

When per-period or per-service logging is enabled the access log file has a specific name generated. Part of this name is the host's name or IP address. By default the host name is used, however if the host IP address is specified the literal address is used, hyphens being substituted for the periods. Accepted values for the [LogNaming] configuration directive are:

- ADDRESS
- HOST
- NAME (default)

Examples of generated per-service (non-per-period) log names:

```
HT_LOGS:131-185-250-202_80_ACCESS.LOG
HT_LOGS:WASD-DSTO-DEFENCE-GOV-AU_80_ACCESS.LOG
HT_LOGS:WASD_80_ACCESS.LOG
```

Examples of generated per-period (with/without per-service) log names:

```
HT_LOGS:131-185-250-202_80_19971013_ACCESS.LOG
HT_LOGS:WASD-DSTO-DEFENCE-GO_80_19971013_ACCESS.LOG
HT_LOGS:WASD_80_19971013_ACCESS.LOG
```

Examples of generated per-instance (per-service and per-period) log names:

```
HT_LOGS:131-185-250-202_80_ACCESS.LOG_KLAATU_HTTPD-80
HT_LOGS:WASD-DSTO-DEFENCE-GOV-AU_80_ACCESS.LOG_KLAATU_HTTPD-80
HT_LOGS:WASD_80_ACCESS.LOG_KLAATU_HTTPD-80
HT_LOGS:131-185-250-202_80_19971013_ACCESS.LOG_KLAATU_HTTPD-80
HT_LOGS:WASD-DSTO-DEFENCE-GO_80_19971013_ACCESS.LOG_KLAATU_HTTPD-80
HT_LOGS:WASD_80_19971013_ACCESS.LOG_KLAATU_HTTPD-80
```

6.11.6 Access Tracking

The term *access tracking* describes the ability to follow a single user's accesses through a particular site or group of related sites. This is accomplished by setting a unique cookie in a user's browser. This cookie is then sent with all requests to that site. The site detects the cookie's unique identifier, or token, and includes it the access log, allowing the user's route through the site or sites to be reviewed. Note that a browser must have cookies enabled for this mechanism to operate.

WASD access tracking is controlled using the [Track...] directives. The tracking cookie uses an opaque, nineteen character string as the token (e.g. ''ORoKJAOef8sAAAkuACc''). This token is spatially and temporally completely unique, generated the first time a user's browser accesses the site. This token is by default added to the server access log in the common format ''remote-ID'' location. It can also be placed into custom logs. From this identifier in the logs a session's progress may be easily tracked. **Note that the token contains nothing related to the user's actual identity!** It is merely a unique identifier that tags a single browser's access trail through a site.

The [Track] directive enables access tracking on a per-server basis. By default all non-proxy services will then have tracking enabled. Individual services may be then be disabled (or enabled in the case of proxy services) using the per-service ''notrack'' and ''track'' parameters.

By default a session track token expires when the user closes the browser. To encourage the browser to keep this token between uses enable multi-session tracking using the [Track-MultiSession] directive. Note that browsers may dispose of any cookie at any time resources become scarce, and that users can also remove them.

Session tracking can be extended from the default of the local server (virtual if applicable) to a group of servers within a local domain. This means the same, initial identifier appears in the logs of all WASD servers in a related group of hosts. Of course tracking must be enabled on all servers. The host grouping is specified using the [TrackDomain] directive (this follows the general rules governing cookie domain behaviour - see RFC2109). Most host grouping require **a minimum of three dots** in the specification. For example (note the leading dot)

```
.site.org.domain
```

which would match the following servers, ''curly.site.org.domain'', ''larry.site.org.domain'', ''moe.site.org.domain'', etc. Sites in top-level domains (e.g. ''edu'', ''com'', ''org'') need only specify a minimum of two periods.

6.11.7 Access Alert

It is possible to mark a path as being of specific interest. When this is accessed by a request the server puts a message into the the server process log and perhaps of greater immediate utility the increase in alert hits is detected by HTTPDMON and this (optionally) provides an audible alert allowing immediate attention. This is enabled on a per-path basis using the SET mapping rule. Variations on the basic rule allow some control over when the alert is generated.

ALERT - at the conclusion of the request
ALERT=MAP - immediately after mapping (early)
ALERT=AUTH - when (any) authorization has been performed

ALERT=END - at the conclusion of the request (default)
ALERT=*integer* - see below
NOALERT - suppress alert for this path

The special case ALERT=*integer* allows a path to be alerted if the final response HTTP status is the same as the integer specified (e.g. 501, 404) or within the category specified (599, 499).

6.12 Include File Directive

WASD uses multiple configuration files for a server and it's site, each one providing for a different functional aspect . . . configuration, virtual services, path mapping, authorization, etc. Generally these configuration files are ''flat'', with all required directives included in a single file. This provides a simple and straight-forward approach suitable for most sites and allows for the provision of Server Administration page online configuration of several aspects.

It is also possible to build site configurations by including the contents of referenced files. This may provide a structure and flexibility not possible using the flat-file approach. All WASD configuration files allow the use of an [IncludeFile] directive. This takes a VMS file specification parameter. The file's contents are then loaded and processed as if part of the parent configuration file. These included files are allowed to be nested to a depth of two (i.e. the configuration file can include a file which may then include another file).

The following is an example used to build up the mapping rules for four virtual services supported on the one server.

```
# HTTPD$MAP

[[alpha.site.com]]
[IncludeFile]  HT_ROOT:[LOCAL]MAP_ALPHA_80.CONF
[[alpha.site.com:443]]
[IncludeFile]  HT_ROOT:[LOCAL]MAP_ALPHA_443.CONF

[[beta.site.com]]
[IncludeFile]  HT_ROOT:[LOCAL]MAP_BETA_80.CONF
[[beta.site.com:443]]
[IncludeFile]  HT_ROOT:[LOCAL]MAP_BETA_443.CONF

[[*]]
[IncludeFile]  HT_ROOT:[LOCAL]MAP_COMMON.CONF
```

Note

Such configurations cannot be managed using Server Administration page interfaces. Files containing [IncludeFile] directives are noted during server startup and if an Server Administration page configuration interface is accessed where this would be a problem an explanatory message and warning is provided. A configuration *can still be saved* but the resulting configuration will be a flat-file representation of the server configuration, not the original hierarchical one.

Chapter 7
"Securing" The Site

This section does not pretend to be a complete guide to keeping the ''bad guys'' out. It does provide a short guide to making a site more-or-less liberal in the way the server supplies information about the site and itself. The reader is also strongly recommended to a number of hard copy and Web based resources on this topic.

The WASD package had it's genesis in making the VMS operating system and associated resources, in a development environment, available via Web technology. For this reason configurations can be made fairly liberal, providing information of use in a technical environment, but that may be superfluous or less-than-desirable in other, possibly commercial environments. For instance, directory listings can contain VMS file system META information, error reports can be generated with similar references along with reporting source code module and line information.

The example configuration files contain a fairly restrictive set of directives. When relaxing these recommendations keep in mind that the more information available about the underlying structure of the site the more potential for subversion. Do not enable functionality that contributes nothing to the fundamental usefulness of the site, or that has the real potential to compromise any given site. This section refers to configuration directives discussed in more detail in later chapters.

It is established wisdom that the only secure computing system is one with no users and no access, that system security is inversely proportional to system usability, and that making something idiot-proof results in only idiots using it. So there are some trade-offs but . . .

don't think it can't happen to you!

A systematic investigation of installed WASD packages by well-known IT professional Jean-loup Gailly during September 2002 revealed a couple of significant implementation flaws which compounded by notable instances of sloppy management practices on two public sites resulted in site compromise (one was mine).

- HT_ROOT:[DOC.MISC]WASD_ADVISORY_020925.TXT
- http://online.securityfocus.com/archive/1/293229

This research has resulted in these server flaws being closed and package security considerations being extensively reviewed. As a result WASD v8.1 is much more resistent to such penetration than previous releases (and slightly less easy to use, but that's one of those trade-offs). My assessment would be that if Gailly did not find it then it wasn't there to find!

Of course any given site's security is a function of the underlying package's security profile, with the site's implementation of that, AND other considerations such as local authorization and script implementations. Pay particular and ongoing attention to site security and integrity.

7.1 Recommended Package Security

The following table provides recommended file protection settings for package top-level directories. Subdirectories share their parents' settings. The package tree is owned by the SYSTEM account. Directories with world READ access have no ACLs. Other directories, not accessable to the world, but sometimes having other degress of access to one or more accounts always have rights identifiers (see below) and associated ACLs to control directory access, and to propagate required access to files created beneath them. The server selectively enables SYSPRV to provide access to some of these areas (e.g. for log creation).

Some pre-version 8.1 directories are not included in this table. These are not significant in versions from 8.1 onwards and may be deleted. They can continue to exist however and the security procedures described below ensure that they comply to the general post-8.1 security model. The file access permissions indicated below are for directory contents. The directory files themselves have settings appropriate for content access.

Package Access

Directory	Access World	Access Other	Description
[AXP-BIN]	none	script:RE	Alpha executable script files
[AXP]	none	none	Alpha build and utility area
[CGI-BIN]	none	script:RE	architecture-neutral script files
[DOC]	read	(world)	package documentation
[EXAMPLE]	read	(world)	package examples
[EXERCISE]	read	(world)	package test files
[HTTP$NOBODY]	none	script:RWED	scripting account default home area
[HTTP$SERVER]	none	server:RWED	server account default home area
[IA64-BIN]	none	script:RE	IA64 executable script files
[IA64]	none	none	IA64 build and utility area

Directory	Access World	Access Other	Description
[INSTALL]	read	(world)	installation, update and secuity procedures
[LOCAL]	none	none	site configuration files
[LOG]	none	none	site access logs
[LOG_SERVER]	none	server:RWED	server process (SYS$OUTPUT) logs
[RUNTIME]	read	(world)	graphics, help files, etc.
[SCRATCH]	none	script:RWED	working file space for scripts
[SCRIPT]	none	none	example architecture-neutral scripts
[SRC]	none	(world)	package source files
[STARTUP]	none	server:RE	package startup procedures
[VAX-BIN]	none	script:RE	VAX executable script files
[VAX]	none	none	VAX build and utility area

It is recommended site-specific directories have settings applied appropriate to their function in comparison to similar package directories. See below for tools to assist in this.

Three rights identifiers provide selective access control to the directory tree. Identifiers were used to allow maximum flexibility for a site in allowing required accounts access to either execute the server or execute scripts. Non-default account names only need to be granted one of these identifiers to be provided with that role's access. Installation, update and/or security utilities create and maintain these identifiers appropriately.

Rights Identifiers

Identifier	Description
WASD_HTTP_SERVER	Indicates the default server account.
WASD_HTTP_NOBODY	Indicates the default scripting account.
WASD_IGNORE_THIS	Looked for by the SECHAN utility to avoid it changing security on site-specific files.

These rights identifiers are applied to directories and files to provide the required level of access. The following example shows the security setting of the top-level CGI-BIN.DIR and one of it content files.

```
$ DIRECTORY /SECURITY CGI-BIN.DIR

Directory HT_ROOT:[000000]

CGI-BIN.DIR;1        [SYSTEM]                          (RWED,RWED,,)
          (IDENTIFIER=WASD_HTTP_SERVER,ACCESS=EXECUTE)
          (IDENTIFIER=WASD_HTTP_NOBODY,ACCESS=EXECUTE)
          (IDENTIFIER=*,ACCESS=NONE)
          (IDENTIFIER=WASD_HTTP_NOBODY,OPTIONS=DEFAULT,ACCESS=READ+EXECUTE)
          (IDENTIFIER=*,OPTIONS=DEFAULT,ACCESS=NONE)
          (DEFAULT_PROTECTION,SYSTEM:RWED,OWNER:RWED,GROUP:,WORLD:)

Total of 1 file.
$ DIRECTORY /SECURITY [CGI-BIN]CGI_SYMBOLS.COM

Directory HT_ROOT:[CGI-BIN]

CGI_SYMBOLS.COM;1    [SYSTEM]                          (RWED,RWED,,)
          (IDENTIFIER=WASD_HTTP_NOBODY,ACCESS=READ+EXECUTE)
          (IDENTIFIER=*,ACCESS=NONE)

Total of 1 file.
```

7.2 Maintaining Package Security

As noted above, WASD version 8.1 and later is much more conservative in what it makes generally available from the package tree, and a site administrator now has to take extraordinary measures to open up certain sections, making it a much more difficult and deliberate action. The package installation, update and security procedures and their associated utilities should always be used to ensure that the installed package continues to conform to it's security baseline.

Package security may be ''refreshed'' or reapplied at any time, and this should be done periodically to ensure that an installed package has not inadvertantly been opened to access where it shouldn't have. Of course this is not a guarantee that any given site is secure. Site security is a function of many factors; package vulnerabilities, site configuration, deployed scripts, cracker determination and expertise, etc., etc. What refreshing the security baseline does is provide a known secure (and WASD-community scrutinized) starting point. It should be used as part of a well considered site security maintenance program.

SECURE.COM

The following DCL procedure resets the package security baseline.

```
$ @HT_ROOT:[INSTALL]SECURE.COM
```

It guides the administrator through a number of stages

- introductory notes
- server account
- scripting account
- package tree security settings

of which each one may be declined. After all of these steps it searches for and executes if found the DCL procedure HT_ROOT:[INSTALL]SECURE.COM. The intent of this file is to allow a site to automatically update any site-specific security settings (and of course modify any set by the main procedure).

SECHAN Utility

The SECHAN utility (pronounced ‘‘session’’) is used by SECURE.COM and its associated procedures to make file system security settings. It is also available for direct use by the site administrator (Section 23.11).

One of the more useful functions of SECHAN is applied using the /IGNORE qualifier.

- **/IGNORE -** It adds an ACE containing the rights identifier WASD_IGNORE_THIS to the target file(s) which results in security settings not being applied in the future. When applying settings the SECHAN utility first checks whether a file has this ACE and if so ignores the file. This is an effective method for isolating site-specific settings from changes by this utility.

  ```
  $ SECHAN /IGNORE HT_ROOT:[CGI-BIN]MY_SCRIPT.COM
  $ SECHAN /IGNORE HT_ROOT:[LOCAL]*.DAT
  $ SECHAN /IGNORE WEB:[DATA...]*.*
  $ SECHAN /IGNORE WEB:[000000]DATA.DIR
  ```

  This ACE can be removed from a file (leaving other entries of any ACL intact) using the /NOIGNORE qualifier. This returns the file(s) subject again to the SECHAN utility.

  ```
  $ SECHAN /NOIGNORE HT_ROOT:[CGI-BIN]MY_SCRIPT.COM
  $ SECHAN /NOIGNORE HT_ROOT:[LOCAL]*.DAT
  ```

- **/ALL -** This overrides the default behaviour of ignoring files that have been tagged using the /IGNORE qualifier. It causes the setting to be applied to ALL files.

Other functionality may prove useful when applied to local parts of the package or web structure.

- **/PACKAGE -** Used alone this qualifier results in the entire HT_ROOT:[000000...] tree being traversed and the default package security settings applied to all package files. Top-level directories that the utility does not recognise as belonging to the package are ignored.

  ```
  $ SECHAN /PACKAGE
  $ SECHAN /PACKAGE /ALL
  ```

- **/ASIF=<name> -** Set the supplied file specification as if it was the specified, top-level WASD directory. This allows a site-specific directory to have the same security settings applied as the specified WASD package directory.

  ```
  $ SECHAN /ASIF=LOCAL WEB:[DATA...]*.*
  $ SECHAN /ASIF=LOCAL WEB:[000000]DATA.DIR
  $ SECHAN /ASIF=CGI-BIN WEB:[SCRIPTS]*.*
  $ SECHAN /ASIF=CGI-BIN WEB:[000000]SCRIPTS.DIR
  $ SECHAN /ASIF=DOC WEB:[HTML...]*.*
  $ SECHAN /ASIF=DOC WEB:[000000]HTML.DIR
  ```

- **/NOSCRIPT -** Modifies the default behaviour of the /PACKAGE qualifier. This changes the default rights identifiers applied to ACEs on files in the [CGI-BIN] and [AXP-BIN]/[VAX-BIN] directories to disallow scripting until manually changed by site administration.

```
$ SECHAN /PACKAGE /NOSCRIPT
```

This section provides only a basic description. More detail may be found in the prologue to the source code.

7.3 Independent Package and Local Resources

Not only does it make it easier to manage site content but is also good security practice to keep server package and site content completely separate (Section 6.1).

This can also be applied to scripts, both source and build areas. Keep your business logic out of the package source tree and potentially prying eyes. The script executables themselves *can* be placed into the package scripting directories but should be built independently from these and copied using locally maintained DCL procedures from build into scripting areas (the HT_ROOT:[INSTALL]SECURE.COM procedures described above may be useful here).

7.4 Configuration

Various configuration and mapping directives can be used to make the site environment more or less liberal in the information it implicitly can provide.

7.4.1 Directory Listings

Published guidelines for securing a Web site generally advise against automatic directory listing generation. Where a home page is not available this may leak information on other directory contents, provide parent and child directory access, etc. Compounding this is the WASD facility to *force* a listing by providing a directory URL with file wildcards (not to decry it's usefulness in some environments).

- **[DirAccess] -** Make ''disabled'' to completely remove the ability to generate directory listings under any circumstances. Setting to ''selective'' means a directory listing is **only** available if the directory contains a file named .WWW_BROWSABLE. When made ''enabled'' a directory listing may be produced anytime it contains no home (welcome) page.
- **[DirWildcard] -** Make ''disabled'' so that requests cannot **force** a directory listing by supplying a URL containing a wildcard file part (when enabled this is provided regardless of whether a home page exists or not).
- **[DirMetaInfo] -** Make ''disabled'' to prevent directory listing pages contain as HTML <META> tags information about the directory, most significantly the VMS file specification for the URL path!

The mapping rule ''SET DIR=*keyword*'' can be used to change this on a per-path basis (Section 13.4.5).

Conservative recommendation: Set ''[DirAccess] selective'' allowing listing for directories containing a file named ''.WWW_BROWSABLE'', disable [DirMetaInfo] and [DirWildcard].

7.4.2 Server Reports

Reports are pages generated by the server, usually to indicate an error or other non-success condition, but sometimes to indicate success (e.g. after a successful file upload). Reports provide either basic or detailed information about the situation. Sometimes the detailed information includes VMS file system details, system status codes etc. To limit this information to a minimum indication adjust the following directives.

- **[ReportBasicOnly] -** Make ''enabled'' to limit the quantity of information to the minimum required to advise of the situation. Such reports give only the HTTP status code and brief explanation of the code's meaning. Note that this can also be done on a per-path basis using mapping rules.
- **[ReportMetaInfo] -** Make ''disabled'' to exclude information on the server software, source code module and line number initiating the report. META information may also contain VMS file or system specific information.
- **[ServerSignature] -** Make ''disabled'' to prevent the inclusion of server software, host and port information as a footer to a report.

The mapping rule ''SET REPORT=*keyword*'' can be used to change some of these on a per-path basis (Section 13.4.5).

Conservative recommendation: Provide minimal error information by enabling [ReportBasicOnly] and disabling [ReportMetaInfo]. Enable [ServerSignature] to provide a slightly more friendly report (server software can easily be obtained from the response header anyway).

7.4.3 Scripting

If a static site is all that's required this source of compromise can simply be avoided.

- **[Scripting] -** Setting this to ''disabled'' prevents all scripting entirely. This includes subprocess CGI and CGIplus, DECnet-based OSU and CGI, and SSI subprocess DCL (<–#dcl –>, <–#exec –>, etc.).

Conservative recommendation: Only deploy scripts your site will actually be using. Remove all the files associated with any other scripts. Do not allow obsolete script environments to remain active. Be proactive.

Also see Section 7.5.

7.4.4 Server Side Includes

SSI documents are pages containing special markup directives interpreted by the server and replaced with dynamic content. This can include detail about the server, the file or files making up the document, and can even include DCL commands and procedure activation for supplying content into the page. All this by anyone who can author on the site.

- **[SSI] -** Setting this to ''disabled'' prevents all Server Side Include processing completely.
- **[SSIexec] -** Setting this to ''disabled'' disallows pages from invoking subprocess DCL to supply content for the page. WASD provides a number of levels of this and the reader is refered elsewhere in this and other documents for further infromation of what can and cannot be done, and by whom, in these subprocesses.

The mapping rule ''SET SSI=*keyword*'' can be used to change some of this on a per-path basis (Section 13.4.5).

Conservative recommendation: Disable [SsiExec].

7.5 Scripting

Scripting has been a notorious source of server compromise, particularly within Unix environments where script process shell command-line issues require special attention. The WASD CGI scripting interface does not pass any arguments on the command line, and is careful not to allow substitution when constructing it's CGI environment. Never-the-less, script behaviours cannot be guaranteed and care should be exercised in their deployment (ask me!)

It is strongly recommended to execute scripts in an account distinct from that executing the server. This should also mean that the accounts are not members of the same group nor should it be a member of any other group. This minimises the risk of both unintentional and malicious interference with server operation through either Inter-Process Communication (IPC) or scripts manipulating files used by the server. The PERSONA facility can be used to further differentiate script activities. See ''Scripting Overview'' for further detail.

The default WASD installation creates two such accounts, with distinct UICs, usernames and home directory space. Nothing should be assumed or read into the scripting account username - it's just a username.

Default Accounts

Username	Description
HTTP$SERVER	Server Account
HTTP$NOBODY	Scripting Account

During startup the server checks for the existence of the default scripting account and automatically configures itself to use this for scripting. If it is not present it falls-back to using the server account. Other account names can be used if the startup procedures are modified accordingly. The default scripting username may be overridden using the /SCRIPT=AS=<username> qualifier (also see the ''Scripting Overview'').

7.6 Authorization

Authorization issues imply controlling access to various resources and actions and therefore require careful planning and implementation if compromise is to be avoided. WASD has a quite capable and versatile authorization and authentication environment, with a significant number of considerations. The reader refered to the chapter on this topic, Chapter 15.

WASD authorization cannot be enabled without the administrator configuring at least three resources, and so therefore cannot easily be ''accidentally'' activated. One of these is the addition of a startup qualifier controlling where authentication information may be sourced. Another the server configuration file. The third, mapping paths against authorization configuration.

For sites that may be particularly sensitive about inadvertant access to some resources it is possible to use the authorization configuration file as a type of *cross-check* on the mapping configuration file. The server /AUTHORIZATION=ALL startup qualifier forces all access to be authorized (even if some are marked ‘‘none’’). This means that if something ‘‘escapes’’ via the mapping file it will very likely be ‘‘caught’’ by an absence in the authorization file.

7.7 Miscellaneous Issues

Although it is of limited usefulness because server identity may be deduced from behaviour and other indicators the exact server and version may be obscured by using the otherwise undocumented /SOFTWARE= qualifier to change the server identification string to (basically) whatever the administrator desires. This identification is included as part of all HTTP response headers.

Historically and by default server configuration and authorization sources are contained within the server package tree. There is no reason why they cannot be located anywhere the site prefers. Generally all that is required is a change to logical name definition and server startup.

Package Tree

Version 8.1 and later is much more conservative in what it makes available of the package tree via the server. The package installation, update and security procedures and their associated utilities should always be used to ensure that the installed package continues to conform to it’s security baseline. See Section 7.2.

Furthermore, with many sites there may be little need to access the full, or any of the WASD package tree. A combination of mapping and/or authorization rules can relatively simply block or control access to it. These examples can be easily tailored to suit a site’s specific requirements.

This example shows blocking all access to the /ht_root/ tree, except for documentation, source code, examples and exercise (performance results) areas.

```
# HTTPD$MAP
pass /ht_root/doc/*
pass /ht_root/src/*
pass /ht_root/example/*
pass /ht_root/exercise/*
fail /ht_root/*
```

The next example forbids all access to the package tree unless authorized (the authorization detail would vary according to the site). It also allows modify access for the Server Administration page and to the /ht_root/local/ area.

```
# HTTPD$MAP
pass /ht_root/*
```

```
# HTTPD$AUTH
[WASD_WEB_ADMIN=id]
/httpd/-/admin/* r+w
/ht_root/local/* r+w
/ht_root/* r
```

Be careful!

There are often multiple paths to a single resource. For instance, it is of little significance blocking access to say /ht_root/doc/ if it's also possible to access it via /doc/.

The following example shows how this might occur.

```
# HTTPD$MAP
fail /ht_root/doc/*
pass /* /ht_root/*
```

Authorization rules can be used to effectively block access to any VMS file specification (it cannot be done during mapping because the translation from path to file system is not performed until mapping is complete).

```
# HTTPD$AUTH
if (path-translated:HT_ROOT:[DOC]*) * none
```

or to selectively allow access

```
# HTTPD$AUTH
[[WASD_VMS_RW=id]]
if (path-translated:HT_ROOT:[DOC]*) * read
```

7.8 Site Attacks

This is not a treatise on Web security and the author is not a security specialist. This is some general advice based on observation. There is little one can do at the server itself to reduce a concerted attack against a site. Common objectives of such attacks include the following (not an exhaustive list).

Platform Vulnerabilities

Where a general attack is launched directed against a specific platform (a combination of operating system and Web server software). Often these can be due to wide-spread infection of systems, meaning many attacks are being launched from a large number of systems (often without the system owners' knowlege or cooperation).

WASD, and OpenVMS in particular, are generally immune to such attacks because they are not Microsoft or Unix based. The impact of the attack becomes one of the nuisance-value traffic as the site is probed by the (sometimes very large number of) source systems.

Site Vulnerabilities

Where a specific attack is made against a site in an attempt to exploit a known vulnerability associated with that platform or environment.

These are perhaps the most worrying, although the *security-by-obscurity* element works in favour of WASD and OpenVMS in this case. Neither are as common as other platforms and therefore do not receive as much attention.

Denial of Service

(DOS) Usually comprise flooding a site with requests in an effort to consume all available network or server resources making it unavailable for legitimate use.

These can be insidious, flooding network equipment as well as systems. Attempts at control are best undertaken at the periphery of the network (routers) although concerted attacks can succeed against the best prepared network.

Password Cracking

Where a systematic attempt to break into one or more accounts is undertaken. These are often repeated, dictionary-based password-guessing attacks.

WASD's authentication functionality notes successive password validation failures and after a reasonable number disables all access via the username for a constantly extended period. Passwords stop being checked and so a dictionary-based attack cannot succeed. Password validation failures can be recorded via OPCOM.

Authorization Holes

Knowing of or searching for resources that should be controlled by authorization but are not.

WASD's /AUTHORIZATION=ALL functionality may assist here (Section 7.6).

Strategies

There are a few strategies for reducing the load on a server experiencing a generalized attack or probing. These can also be used to ''discourage'' the source from considering the site an easy target. Unfortunately most require request acceptance and at least some processing before taking action. The general idea is to identify either the source site or some characteristic of the request that indicates it could not possibly be legitimate. Most platform-specific attacks have such a signature. For instance attacks against Microsoft platforms often involve probes for backdoors into non-server executables. These can be identified by the path containing strings such as ''/winnt/'', ''/system32/'', ''/cmd.exe'' or variations on them. This style will be used in examples below.

- If the source IP address is known then the [Reject] (and/or [Accept]) configuration directives can be used to reject the request connection very early in the processing. The source agent receives a message about access being rejected.

```
[Reject]
131.185.250.*
the.host.name
```

- Mapping rules in combination with conditionals may be used to redirect the request. This redirection could be to another, non-existent site, in the hope that the source agent will use the supplied URL and thus divert some activity away from the local site.

```
if (remote-host:the.host.name)
   redirect * http://the.host.name/*
endif

redirect **/winnt/** http://does.not.exist/
```

- Mapping rule redirection can also be used to just ‘‘drop’’ the connection without any further interaction or processing. The source agent receives no response, just a broken connection.

```
if (remote-addr:131.185.250.*)
   pass * "000 just drop it!"
endif

pass **/system32/** "000 just drop it!"
```

- The *hiss* facility (Section 23.2) returns a stream of random alpha-numeric characters (a sort of *white-noise*). No response header is provided. Such a response might cause the source agent at best some distress (perhaps disabling it) or at least disuade it from continuing with more probes (as the target is obviously not a Web server ;^)

```
if (remote-addr:131.185.250.*) map * /hiss/*
script /hiss/* /hiss/*

map **/cmd.exe** /hiss/*/cmd.exe*
script /hiss/* /hiss/*
```

Chapter 8
Server Configuration Directives

The example configuration file can be used as a template.

online hypertext link

By default, the system-table logical name HTTPD$CONFIG locates a common configuration file, unless a per-server file is specified using a job-table logical name. Simple editing of the configuration file changes the rules. Alternatively the Server Administration page configuration interface may be used.

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 Section 6.12.

Some directives take a single parameter, such as an integer, string or boolean value. Other directives can/must have multiple parameters. The version 4 configuration requires the directive to be placed on a line by itself and each separate parameter on a separate line following it. All parameter lines apply to the most recently encountered directive.

Note that all *boolean* directives are *disabled* (OFF) by default. This is done so that there can be no confusion about what is enabled and disabled by default. To use directive controlled facility it **must** be explicitly enabled.

8.1 Functional Groupings

Authentication/Authorization

[AuthBasic]	enable BASIC method
[AuthCacheEntriesMax]	maximum concurrent authentication cache entries
[AuthCacheEntrySize]	maximum authentication cache entry size in bytes
[AuthCacheMinutes]	minutes before explicitly reauthorizing user from sources
[AuthDigest]	enable DIGEST method

[AuthDigestGetLife]	DIGEST method GET lifetime
[AuthDigestPutLife]	DIGEST method PUT lifetime
[AuthFailureLimit]	retries allowed before username is marked as intruder
[AuthRevalidateLoginCookie]	helps prevent redundant authorization requests when user revalidation is enabled
[AuthRevalidateUserMinutes]	minutes before use needs to reenter password
[AuthSysUafAcceptExpPwd]	accept expired SYSUAF passwords
[AuthSysUafPwdExpURL]	redirection URL is SYSUAF password is expired
[AuthSysUafUseAcme]	use the ACME service for all VMS SYSUAF based authentication

Buffer Sizes

[BufferSizeDclCgiPlusIn]	number of bytes allocated to scripting subprocess CGIPLUSIN mailbox
[BufferSizeDclCommand]	bytes allocated to scripting subprocess SYS$COMMAND mailbox
[BufferSizeDclOutput]	bytes allocated to scripting subprocess SYS$OUTPUT mailbox
[BufferSizeNetRead]	bytes allocated to client request read buffer, and to the scripting subprocess SYS$INPUT mailbox
[BufferSizeNetWrite]	bytes allocated to client output buffer

Content-Type

[AddType]	add a content-type
[AddMimeTypesFile]	add the contents of a standard MIME.TYPES file
[CharsetConvert]	conversion of one character set to another
[CharsetDefault]	default character set for text responses
[StreamLF]	enable and set maximum size of automatic Stream-LF conversion

Directory Listing

[AddIcon]	path to icon for a specified content-type
[AddBlankIcon]	path to blank icon
[AddDefaultIcon]	path to default icon
[AddDirIcon]	path to directory icon
[AddParentIcon]	path to parent icon

[AddUnknownIcon]	path to icon for unknown content-type
[DirAccess]	enable and form of listing
[DirBodyTag]	specify HTML body tag of listing pages
[DirDescriptionLines]	number of HTML file lines searched for document title
[DirLayout]	layout of the various listing components
[DirMetaInfo]	add server and VMS directory information
[DirNoImpliedWildcard]	do not add wildcards to request if not present in path
[DirNoPrivIgnore]	ignore, do not report, privilege violations on files/directories
[DirOwner]	allow owner of file to be included in layout directive
[DirPreExpired]	pre-expire listing responses
[DirReadMeFile]	specify read-me files
[DirWildcard]	allow wildcards to be specified at all

File Cache

[CacheChunkKBytes]	memory block allocation size
[CacheEntriesMax]	maximum number of files allowed in cache
[CacheFileKBytesMax]	maximum size of a file
[CacheFrequentIntervals]	identify active files
[CacheFrequentPeriod]	identify active file
[CacheGuardPeriod]	prevent early reloads
[CacheTotalKBytesMax]	maximum memory to be consumed by cache
[CacheValidatePeriod]	maximum period before the cache checks for file modification

Logging

[Logging]	enable logging
[LogExcludeHosts]	hosts to be excluded from log
[LogExtend]	default allocation/extend in blocks
[LogFile]	provides part or all of log file name
[LogFormat]	nature and layout of log contents
[LogNaming]	how the log name is be constructed
[LogPeriod]	period at which new logs are created

[LogPerInstance]	create a separate log for each instance process
[LogPerService]	create a separate log for each configured service
[LogPerServiceHostOnly]	suppress service port number as component of log name
[Track]	enable session tracking
[TrackMultiSession]	track across sessions
[TrackDomain]	host or hosts it applies to

Operator Console and Log

[OpcomAdmin]	Server Administration directives
[OpcomAuthorization]	authentication/authorization messages, e.g. failures
[OpcomControl]	CLI HTTPd control directives
[OpcomHTTPd]	HTTPd events (e.g. startup, exit, SSL private key password requests)
[OpcomProxyMaint]	proxy file cache maintenance
[OpcomTarget]	target operator for online messages

Miscellaneous

[Accept]	restrictive list of host from which to accept requests
[ActivityDays]	activity graph duration
[Busy]	number of concurrent connections allowed
[DNSLookupClient]	enable client host name lookup
[DNSLookupLifeTime]	host name lookup cache entry lifetime
[DNSLookupRetry]	number two second attempts to resolve client host name
[InstanceMax]	number of per-node server processes to maintain
[Monitor]	enable HTTPDMON data exchange
[Port]	default port
[PutMaxKBytes]	maximum size of a POST or PUT
[PutVersionLimit]	maximum RMS file versions retained in a POST or PUT
[RegEx]	enable regular expression matching
[Reject]	proscriptive list of hosts from which request will be rejected
[RequestHistory]	number of requests kept for request report
[SearchScript]	path to default search script

[SearchScriptExclude]	list of file extensions excluded from implied keyword search
[Service]	list of host names and/or port to create services for
[ServiceNotFoundURL]	redirection URL when a request service is not configured
[Welcome]	list of file names that are checked for as home pages

Proxy Serving

[ProxyCache]	enable proxy caching
[ProxyCacheFileKBytesMax]	maximum size of response for caching
[ProxyCacheDeviceCheckMinutes]	minutes between check of cache device usage
[ProxyCacheDeviceDirOrg]	flat 256 or 64x64 directory organization
[ProxyCacheDeviceMaxPercent]	maximum percentage of cache device used before purge
[ProxyCacheDevicePurgePercent]	during purge reduce by this many percent
[ProxyCacheNoReloadSeconds]	prevent pragma reloads for this period
[ProxyCachePurgeList]	list of file ages used during purge
[ProxyCacheReloadList]	list of file ages before realod from source
[ProxyCacheRoutineHourOfDay]	hour of day routine cache purge occurs
[ProxyForwarded]	add ''Forwarded:'' to requests
[ProxyHostLookupRetryCount]	DNS resolution retry count
[ProxyReportLog]	report failures to process log
[ProxyReportCacheLog]	report cache failures to process log
[ProxyServing]	enable proxy server
[ProxyXForwardedFor]	add ''X-Forwarded-For:'' to requests

Reports

[ErrorReportPath]	path to script, SSI or ''flat'' error document
[ErrorRecommend]	for server generated error include probable cause
[ReportBasicOnly]	only ever generate reports containing basic details
[ReportMetaInfo]	add server information to directory listings, etc.
[ServerAdmin]	email address for server-related contact
[ServerAdminBodyTag]	specify HTML body tag of Server Administration (menu) pages
[ServerReportBodyTag]	specify HTML body tag of error and other report pages

[ServerSignature]	add server information to the foot of error and other report pages

Request Timeout

[TimeoutInput]	period a connection can wait before sending request
[TimeoutKeepAlive]	period a connection is kept active after request conclusion
[TimeoutOutput]	period a response can continue to output
[TimeoutNoProgress]	period a response can continue without data transfer progress

Scripting

[CgiStrictOutput]	script output must be CGI compliant
[DclBitBucketTimeout]	period a script continues after a client prematurely disconnects
[DclCgiPlusLifeTime]	period of non-use before CGIplus subprocess is deleted
[DclCleanupScratchMinutesMax]	maximum minutes between HT_SCRATCH cleanups
[DclCleanupScratchMinutesOld]	cleanup files older than this
[DclDetachProcess]	use detached scripting processes rather than subprocesses
[DclGatewayBG]	enable raw TCP/IP socket for scripts
[DclHardLimit]	maximum number of concurrent subprocesses
[DclScriptRunTime]	script execution environment
[DclSoftLimit]	maximum number of subprocesses before proactive deletion begins
[DclSpawnAuthPriv]	spawn subprocesses with account's authorized privileges
[DclZombieLifeTime]	period of non-use before a CGI/CLI subprocess is deleted
[DECnetReuseLifeTime]	period of non-use before a DECnet process is released
[DECnetConnectListMax]	maximum number of DECnet processes
[Scripting]	enables and disables all scripting

Server Side Includes

[SSI]	enable Server Side Includes (SSI)
[SSIaccesses]	allow access counting
[SSIexec]	allow DCL commands
[SSIsizeMax]	maximum source file size

8.2 Alphabetic Listing

1. **[Accept]** ***host/domain name*** *(default: all)*

   One or more (comma-separated if on the same line) internet host/domain names, with ‘‘*’’ wildcarding for host/subdomain matching, to be explicitly allowed access. If DNS lookup is not enabled hosts must be expressed using literal addresses (see [DNSLookup] directive). Also see the [Reject] directive. Reject directives have precedence over Accept directives. The Accept directive may be used multiple times.

   Examples:

   ```
   [Accept]
   *.wasd.dsto.defence.gov.au
   131.185.250.*
   ```

2. **[ActivityDays]** ***integer*** *(default: 0)*

   Specifies the number of days to record activity statistics, available in report form from the Server Administration page (Section 18.4). Zero disables this data collection. The maximum is 28 days. 11520 bytes per day, and 80640 per week, is required to store the per-minute data.

3. **[AddIcon]** ***icon-URL ALT-text template*** *(no default)*

   Specifies a directory listing icon and alternative text for the mime content type specified in the template.

   Examples:

   ```
   [AddIcon]
   /icon/-/doc.gif    [HTM]  text/html
   /icon/-/text.gif   [TXT]  text/plain
   /icon/-/image.gif  [IMG]  image/gif
   ```

4. **[AddBlankIcon]** ***icon-URL***
   [AddDefaultIcon] ***icon-URL ALT-text***
   [AddDirIcon] ***icon-URL ALT-text***
   [AddParentIcon] ***icon-URL ALT-text***
   [AddUnknownIcon] ***icon-URL ALT-text*** *(no defaults)*

   Specifies a directory listing icon for these non-content-type parts of the listing.

   Examples:

   ```
   [AddBlankIcon]    /icon/-/blank.gif    _____
   [AddDefaultIcon]  /icon/-/file.gif     [FIL]
   [AddDirIcon]      /icon/-/dir.gif      [DIR]
   [AddParentIcon]   /icon/-/back.gif     [<--]
   [AddUnknownIcon]  /icon/-/unknown.gif  [???]
   ```

5. **[AddMimeTypesFile]** ***file specification*** *(no default)*

   Add the content-types of a (de facto) standard MIME.TYPES file to the already configured [AddType] content-types. This binds a file suffix (extension, type) to a MIME content-type. Any specification in this file will supercede any previously defined via [AddType]. A MIME.TYPES file looks something like

```
# MIME type   Extension
application/msword            doc
application/octet-stream      bin dms lha lzh exe class
application/oda               oda
application/pdf               pdf
application/postscript        ai eps ps
application/rtf               rtf
```

The WASD server uses a number of extensions to provide additional information. See Section 6.6.

6. **[AddType]** ***.suffix content-type*** **[*script-name*] [*description*]** *(no default)*

   Binds a file suffix (extension, type) to a mime content type. The script name is used to auto-script against a specified file type. Use a hyphen as a place-holder and to indicate no auto-script. The description is used as documentation for directory listings.

```
[AddType]
.html  text/html   HyperText Markup Language
.txt   text/plain  plain text
.gif   image/gif   image (GIF)
.hlb   text/x-script /Conan  VMS Help library
.decw$book   text/x-script   /HyperReader    Bookreader book
*  internal/x-unknown   application/octet-stream
#*  internal/x-unknown  text/plain
```

   The content-type string may include a specific character set. In this way non-default sets (which is usually ISO-8859-1) can be specified for any particular site or any particular file type. Enclose the content-type string with double-quotation marks.

```
[AddType]
.html    "text/html; charset=ISO-8859-1"   HTML (ISO-8859-1)
.html_5  "text/html; charset=ISO-8859-5"   Cyrillic HTML (ISO-8859-5)
.html_r  "text/html; charset=KOI8-R"       Cyrillic HTML (KOI8-R)
.txt     "text/plain; charset=ISO-8859-1"  plain text (ISO-8859-1)
.txt_5   "text/plain; charset=ISO-8859-5"  Cyrillic text (ISO-8859-5)
.txt_r   "text/plain; charset=KOI8-R"      Cyrillic text (KOI8-R)
```

   To provide additional information for correct handling of FTP transfers the content-type may have an FTP transfer mode indicated. This is provided by appending the mode directly after the content type (allow no white-space). One of three characters is used. An ‘‘A’’ indicates that this file type should be FTP transfered in ASCII mode. An ‘‘I’’ or a ‘‘B’’ indicates that this file type should be FTP transfered in Image (binary) mode. The following example provides the syntax.

```
[AddType]
.ps    application/postscript(ftp:A)   Postscript document
```

7. **[AuthBasic]** ENABLED|DISABLED *(default: DISABLED)*

   Enables or disables BASIC username authentication. See Chapter 15.

8. **[AuthCacheEntrySize]** ***integer*** *(default: 768)*

   Maximum size of an authentication cache entry. The only reason where this may need to be increased is where a site is using the /PROFILE functionality and one or more accounts have a particularly large number of rights identifiers.

9. **[AuthCacheMinutes]** ***integer*** *(default: 60)*

The number of minutes authentication information is cached before being revalidated from the authentication source. Zero disables caching (with a resultant impact on performance as each request requiring authentication is validated directly from the source).

10. **[AuthDigest]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables Digest username authentication. See Chapter 15.

11. **[AuthDigestGetLife]** ***integer*** *(default: 0)*

    The number of seconds a digest nonce for a GET request (read) can be used before becoming stale.

12. **[AuthDigestPutLife]** ***integer*** *(default: 0)*

    The number of seconds a digest nonce for a PUT (/POST/DELETE ... write) request can be used before becoming stale.

13. **[AuthFailureLimit]** ***integer*** *(default: 10)*

    The number of unsuccessful attempts at authentication before the username is disabled. Once disabled any subsequent attempt is automatically refused without further reference to the authentication source. A disabled username can be reenabled by simply purging the cache.

14. **[AuthRevalidateLoginCookie]** ENABLED|DISABLED *(default: DISABLED)*

    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.

15. **[AuthRevalidateUserMinutes]** ***integer*** *(default: 60)*

    The number of minutes between authenticated requests that user authentication remains valid before the user is forced to reenter the authentication information (via browser dialog). Zero disables the requirement for revalidation.

16. **[AuthSysUafAcceptExpPwd]** ENABLED|DISABLED *(default: DISABLED)*

    If a SYSUAF authenticated password has expired (password lifetime has been reached) accept it anyway (in much the same way network logins are accepted in similar circumstances). This is very different to *account expiry*, after which authentication is always rejected.

17. **[AuthSysUafPwdExpURL]** ***string*** *(default: none)*

    If a SYSUAF authenticated password is/has expired the request is redirected to this URL to change the password. See Section 15.14

18. **[AuthSysUafUseAcme]** ENABLED|DISABLED *(default: DISABLED)*

    On applicable platforms (Alpha (and IA64?), OpenVMS 7.3 and later) use the ACME service to perform SYSUAF authentication and SYSUAF password change. The immediate advantage of using ACME is 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.

19. **[BufferSizeDclCgiPlusIn]** ***integer*** *(default: 2048)*

    The number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess CGIPLUSIN mailbox.

20. **[BufferSizeDclCommand]** ***integer*** *(default: 3072)*

    The number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess SYS$COMMAND mailbox.

21. **[BufferSizeDclOutput]** ***integer*** *(default: 4096)*

    The number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess SYS$OUTPUT mailbox.

22. **[BufferSizeNetRead]** ***integer*** *(default: 2048)*

    The number of bytes allocated to the network read buffer (used for request header, POST body, etc.). Also the number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess SYS$INPUT mailbox (allowing a script to read a request body).

23. **[BufferSizeNetWrite]** ***integer*** *(default: 4096)*

    Number of bytes allocated to the network write buffer. This buffer is used as the basic unit when transfering file contents (from cache or the file system), as an output buffer during SSI pocessing, directory listing, etc. During many activities multiple outputs are buffered into this storage before being written to the network.

24. **[Busy]** ***integer*** *(default: 10)*

    The maximum number of concurrent client connections before a ''*server too busy right now ... try again shortly*'' error is returned to the client.

25. **[Cache]** ENABLED|DISABLED *(default: DISABLED)*

    File cache control.

26. **[CacheChunkKBytes]** ***integer*** *(default: 0)*

    Granularity of memory blocks allocated to file data, in kilobytes.

27. **[CacheEntriesMax]** ***integer*** *(default: 0)*

    Maximum number of files loaded into the cache before entries are reused removing the original contents from the cache.

28. **[CacheFileKBytesMax]** ***integer*** *(default: 0)*

    Maximum size of a file before it is not a candidate for being cached, in kilobytes.

29. **[CacheFrequentIntervals]** ***integer*** *(default: 0)*

    Minimum, total number of hits an entry must sustain before being a candidate for [CacheFrequentPeriod] assessment.

30. **[CacheFrequentPeriod]** ***hh:mm:ss*** *(default: 00:00:00)*

    If a file has been hit at least [CacheFrequentIntervals] times in total and the last was within the period here specified it will not be a candidate for reuse. See Chapter 20.

31. **[CacheGuardPeriod]** ***integer*** *(default: 15)*

During this period subsequent *reloads* (no-cache) requests will not result in the entry being revalidated or reloaded. This can guard period can help prevent unnecessary file system activity.

32. **[CacheHashTableEntries]** ***integer*** *(default: 0)*

    Obsolete for WASD V8.0 and following.

33. **[CacheTotalKBytesMax]** ***integer*** *(default: 0)*

    Maximum memory allocated to the cache, in kilobytes.

34. **[CacheValidatePeriod]** ***hh:mm:ss*** *(default: 00:00:00)*

    The interval after which a cache entry's original, content revision time is revalidated against the file's current revision time. If not the same the contents are declared invalid and reloaded.

35. **[CharsetConvert]** ***string*** *(default: none)*

    Document and CGI script output can be dynamically converted from one character set to another using the standard VMS NCS conversion library. This directive provides the server with character set aliases (those that are for all requirements the same) and which NCS conversion function may be used to convert one character set into another. The general format is

    ```
    document-charset  accept-charset[,accept-charset..]  [NCS-function-name]
    ```

    When this directive is configured the server compares each text response's character set (if any) to each of the directive's *document charset* string. If it matches it then compares each of the *accepted charset* (if multiple) to the request ''Accept-Charset:'' list of accepted characters sets. If the same is is either accepted as-is or if a conversion function specified converted by NCS as the document is transfered.

    ```
    windows-1251 windows-1251,cp-1251
    windows-1251 koi8-r koi8r_to_windows1251_to_koi8r
    koi8-r koi8-r,koi8
    koi8-r windows-1251,cp-1251 koi8r_to_windows1251
    ```

36. **[CharsetDefault]** ***string*** *(default: none)*

    The default character set sent in the response header for text documents (plain and HTML). English language sites should specify ISO-8859-1, other Latin alphabet sites, ISO-8859-2, 3, etc. Cyrillic sites might wish to specify ISO-8859-5 or KOI8-R, and so on.

37. **[CgiStrictOutput]** ENABLED|DISABLED *(default: DISABLED)*

    A script must output a full HTTP or CGI-compliant response. If a plain-text stream is output an error is reported (being the more common behaviour for servers). Errors in output can be disagnosed using the WATCH facility.

38. **[DclBitBucketTimeout]** ***hh:mm:ss*** *(default: 0)*

    Period a script is allowed to continue processing before being terminated after a client prematurely disconnects. An approptiate setting allows most scripts to conclude elegantly and be available for further use. This improves scripting efficiency significantly. Setting this period to zero terminates scripts (and their associated processes) immediately a client is detected as having disconnected.

39. **[DclCleanupScratchMinutesMax] *integer*** *(default: 0)*

    Whenever the last scripting process is removed from the system, or this number of minutes maximum (whichever occurs first), scan the HT_SCRATCH directory (if logical defined and it exists) deleting all files that are older than [DclCleanupScratchMinutesOld] minutes. Setting to zero disables HT_SCRATCH scans.

40. **[DclCleanupScratchMinutesOld] *integer*** *(default: 0)*

    When performing a [DclCleanupScratchMinutesMax] scan delete files that are older than this value (or the value specified by [DclCleanupScratchMinutesMax], whichever is the larger).

41. **[DclCgiPlusLifeTime] *hh:mm:ss*** *(default: 0)*

    If this value is zero CGIplus subprocess may persist indefinitely (excluding explicit and proactive server purging). If non-zero the CGIplus subprocess is terminated the specified period after it last processed a request. This helps prevent sporadically used scripts from clogging up a system.

42. **[DclDetachProcess]** ENABLED|DISABLED *(default: DISABLED)*

    By default scripts are executed within server subprocesses. When enabled this instructs the server to create detached processes. This side-steps the issues of having pooled process quotas and also allows non-server-account scripting (Section 5.5, User Account Scripting and in particular ‘‘Scripting Overview, Introduction’’.

43. **[DclDetachProcessPriority] *integer[,integer]*** *(default: same as server)*

    When detached scripting processes are created it is possible to assign them base priorities lower that the server itself. This directive takes one or two (comma-separated) integers that determine how many priorities lower than the server scripting processes are created. The first integer determines server processes. A second, if supplied, determines user scripts. User scripts may never be a higher priority that server scripts.

    ```
    [DclDetachProcessPriority]  1
    [DclDetachProcessPriority]  0,1
    [DclDetachProcessPriority]  1,2
    ```

    The first of these examples would set both server and user script processes one below the server process. The second, server scripts at the same priority and user scripts one below. The last, server scripts one below, and user scripts two below.

44. **[DclGatewayBG]** ENABLED|DISABLED *(default: DISABLED)*

    When enabled, non-SSL, subprocess script CGI environments have a CGI variable WWW_GATEWAY_BG created containing the device name (BG*nnnn*:) of the TCP/IP socket connected to the client. This socket may be accessed by the script for transmission of data directly to the script bypassing the server entirely. This is obviously much more efficient for certain classes of script. For purposes of accurate logging the server does need to be informed of the quantity of data transfered using a CGI callout. See ‘‘Scripting Environment’’ document.

45. **[DclHardLimit] *integer*** *(default: 0)*

    The maximum number of DCL/CGI script processing subprocesses that may ever exist concurrently (works in conjunction with [DclSoftLimit].

46. **[DclScriptRunTime]** ***string*** *(default: none)*

    One or more file type (extension) specification and scripting verb pairs. See ''Scripting Overview, Runtime''.

47. **[DclSoftLimit]** ***integer*** *(default: 0)*

    The number of DCL/CGI script processing subprocesses after which idle subprocesses are deleted to make room for new ones. The [DclHardLimit] should be approximately 25% more than the [DclSoftLimit]. The margin exists to allow for occasional slow run-down of deleted/finishing subprocesses. If these limits are not set (i.e. zero) they are calculated with [Busy] using ''[DclSoftLimit] = [Busy]'' and ''[DclHardLimit] = [DclSoftLimit] + [DclSoftLimit] / 4''.

48. **[DclSpawnAuthPriv]** ENABLED|DISABLED *(default: DISABLED)*

    By default, when a DCL/scripting subprocess is spawned it inherits the server's currently enabled privileges, which are **none**, not even TMPMBX or NETMBX. If this parameter is enabled the subprocess is created with the server account's SYSUAF-authorized privileges (which should never be other than NETMBX and TMPMBX). Use with caution.

49. **[DclZombieLifeTime]** ***hh:mm:ss*** *(default: 00:00:00)*

    If this value is zero the use of persistant DCL subprocesses is disabled. If non-zero the *zombie* subprocess is terminated the specified period after it last processed a request. This helps prevent zombie processes from clogging up a system. See ''Scripting Environment'' document.

50. **[DECnetReuseLifeTime]** ***hh:mm:ss*** *(default: 00:00:00)*

    Period a DECnet scripting connection is maintained with the network task. Zero disables connection reuse.

51. **[DECnetConnectListMax]** ***integer*** *(default: 0)*

    The size of the list used to manage connections for DECnet scripting. Zero effectively allows the server to use as many DECnet scripting connections as demanded.

52. **[DirAccess]** ENABLED|DISABLED|SELECTIVE *(default: DISABLED)*

    Controls directory listings. SELECTIVE allows access only to those directories containing a file .WWW_BROWSABLE. The WASD HTTPd directory access facility always ignores directories containing a file named .WWW_HIDDEN. Also see the [DirWildcard] directive.

53. **[DirBodyTag]** ***string*** *(default: <BODY>)*

    Specifies the HTML <BODY> tag for directory listing pages. This allows some measure of site ''look-and-feel'' in page colour, background, etc. to be employed.

54. **[DirDescriptionLines]** ***integer*** *(default: 0)*

    Non-Zero enables HTML file descriptions during listings. Generating HTML descriptions involves opening each HTML file and searching for <TITLE>...</TITLE> and <H1>...</H1> text to generate the description. This is an obviously resource-intensive activity and on busy servers or systems may be disabled. Any non-zero number specifies the number of lines to be searched before quitting. Set to a very high number to search all of files' contents (e.g. 999999).

55. **[DirLayout]** ***string*** *(default: I_ _L_ _R_ _S_ _D)*

Allows specification of the directory listing layout. This is a short, case-insensitive string that specifies the included fields, relative placement and optionally the width of the fields in a directory listing. Each field is controlled by a single letter and optional leading decimal number specifying its width. If a width is not specified an appropriate default applies. An underscore is used to indicate a single space and is used to separate the fields (two consecutive works well).

C - creation date
D - description (generally best specified last)
D:L - for files, make a link out of the description text
I - icon (takes no field-width attribute)
L - link (highlighted anchor using the name of the file)
L:F - file-system name (for ODS-5 displays spaces, etc.)
L:N - name-only, do not display the extension
L:U - force name to upper-case
N - name (no link, why bother? who knows!)
O - owner (can be disabled)
R - revision date
S - size
S:B - in bytes (comma-formatted)
S:D - decimal kilos (see below)
S:F - kilo and mega are displayed to one decimal place
S:K - in kilo-bytes (and fractions thereof)
S:M - in mega-bytes (and fractions thereof)
U - upper-case file and directory names (must be the first character)

The following shows some examples:

```
[DirLayout]       I__L__R__S__D
[DirLayout]       I__L__R__S:b__D
[DirLayout]       I__15L__S__D
[DirLayout]       UI__15L__S__D
[DirLayout]       15L__9R__S
[DirLayout]       15N_9C_9R_S
[DirLayout]       I__L__R__S:d__D
[DirLayout]       25D:l__S:b__C__R
```

The size of files is displayed by default as 1024 byte kilos. When using the ''S:k'', ''S:m'' and ''S:f'' size modifiers the size is displayed as 1000 byte kilos. If it is prefered to have the default display in 1000 byte kilos then set the directory listing layout using:

```
[DirLayout]       I__L__R__S:d__D
```

If unsure of the kilo value being used check the ''<META>'' information in the directory listing.

56. **[DirMetaInfo]** ENABLED|DISABLED *(default: DISABLED)*

Includes, as <META> information, the software ID of the server and any relevant VMS file information.

57. **[DirNoImpliedWildcard]** ENABLED|DISABLED *(default: DISABLED)*

When a directory is accessed having no file or type component and there is no welcome page available a directory listing is generated. By default any other directory accessed from this listing has the implied wildcards "*.*" added, consequently forcing directory listings. If enabled, this directive ensures no wildcards are added, so subsequent directories accessed with welcome pages display the pages, not a forced listing.

58. **[DirNoPrivIgnore]** ENABLED|DISABLED *(default: DISABLED)*

To prevent browsing through directories (perhaps due to inadvertant mapping) that have file permissions allowing no WORLD access the server stops listing and reports the error the first time a protection violation occurs. This behaviour may be changed to ignore the violation, listing only those files to which it has access.

59. **[DirOwner]** ENABLED|DISABLED *(default: DISABLED)*

Allows specification and display of the RMS file owner information.

60. **[DirPreExpired]** ENABLED|DISABLED *(default: DISABLED)*

Directory listings and trees may be *pre-expired*. That is, the listing is reloaded each time the page is referenced. This is convenient in some environments where directory contents change frequently, but adds considerable over-head and so is disabled by default. Individual directory listings may have the default behaviour over-ridden using syntax similar to the following examples:

```
/dir1/dir2/*.*?httpd=index?expired=yes
/dir1/dir2/*.*?httpd=index?expired=no
/tree/dir2/?httpd=index?expired=yes
/tree/dir1/dir2/?httpd=index?expired=no
```

61. **[DirReadme]** TOP|BOTTOM | OFF *(default: DISABLED)*

If any of the files provided using the [DirReadMeFile] directive are located in the directory the contents are included at the top or bottom of the listing (or not at all). Plain-text are included as plain-text, HTML are included as HTML allowing markup tags to be employed.

62. **[DirReadMeFile]** FILE.SUFFIX *(no default)*

Specifies the names and order in which a directory is checked for *read-me* files. This can be enabled or disabled using the [DirReadme] directive. Plain-text are included as plain-text, HTML are included as HTML allowing markup tags to be employed.

Examples:

```
[DirReadMeFile]
readme.html
readme.htm
readme.
readme.txt
readme.1st
```

63. **[DirWildcard]** OFF|ON *(default: DISABLED)*

This enables the facility to *force* the server to provide a directory listing by providing a wildcard file specification, even if there is a home (welcome) document in the directory. This should not be confused with the [DirAccess] directive which controls directory listing itself.

64. **[DNSLookupClient]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables connection request host name resolution. This functionality may be expensive (in terms of processing overhead) and make serving granularity coarser if DNS is involved. If not enabled and logging is, the entry is logged against the literal internet address. If not enabled any [Accept], [Reject] or conditional directive, etc., must be expressed as a literal address.

65. **[DNSLookupLifetime]** ***hh:mm:ss*** *default 00:10:00*

    The period for which a host name/address is cached (applies to both client lookup and proxy host lookup).

66. **[DNSLookupRetry]** ***integer*** *(default: 2)*

    The number of attempts, at two second intervals, made to resolve a host name/address (applies to both client lookup and proxy host lookup).

67. **[ErrorReportPath]** ***string [status...]*** *(default: none)*

    Specifies the **URL-format path** to an optional, error reporting SSI document or script. See Section 6.9. This path can subsequently be remapped during request processing. Optional, space-separated HTTP status codes restrict the path to those codes, with the remainder handled by server-internal reporting.

68. **[ErrorRecommend]** ENABLED|DISABLED *(default: DISABLED)*

    Provides a short message recommending action when reporting an error to a client. For example, if a document cannot be found it may say:

    ```
    (document, or bookmark, requires revision)
    ```

69. **[InstanceMax]** ***integer*****|**CPU *(default: 1)*

    Number of per-node server processes to create and maintain. If set to ''CPU'' once instance per CPU is created.

70. **[Logging]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables the request log. Logging can slow down request processing and adds overhead. The log file name must be specified using the /LOG qualifier or HTTPD$LOG logical name (Logical Names.

71. **[LogExcludeHosts]** ***string*** *(default: none)*

    One or more (comma-separated if on the same line) internet host/domain names, with ''*'' wildcarding for host/subdomain matching, requests from which are not placed in any log files. If DNS lookup is not enabled hosts must be expressed using literal addresses (see [DNSLookup] directive). Use for excluding local or web-maintainer's host from logs.

    Example:

    ```
    [LogExcludeHosts]
    *.wasd.dsto.defence.gov.au
    131.185.250.*
    ```

72. **[LogExtend]** ***integer*** *(default: 0)*

Number of blocks allocated when when a log file is opened or extended. If set to zero it uses the process default (SET RMS_DEFAULT /EXTEND_QUANTITY).

73. **[LogFile]** ***string*** *(default: none)*

    Provides some or all of the access log file name. See Section 6.11.2.

74. **[LogFormat]** ***string*** *(default: COMMON)*

    Specifies one of three pre-defined formats, or a user-definable format. See Section 6.11.1.

75. **[LogGlobal]** ***integer*** *(default: 0)*

    Number of global buffers to use when multiple instances are configured. This directive improves performance (by delaying write-to-disk) of multi-instance configurations by using RMS global buffering for shared write access to log files. As each log file has a multiblock count of 127 this is probably best specified as 1 or some other small number. It consumes global sections and global pages from system-wide resources.

76. **[LogNaming]** ***string*** *(default: none)*

    When [LogPeriod] or [LogPerService] directives are used to generate multiple log files this directive may be used to modify the naming of the file. See Section 6.11.5.

77. **[LogPeriod]** ***string*** *(default: none)*

    Specifies a period at which the log file is changed. See Section 6.11.2.

78. **[LogPerInstance]** ENABLED|DISABLED *(default: DISABLED)*

    When multiple instances are configured (Section 6.2) create a separate log for each. This has significant performance advantages. See Section 6.11.4.

79. **[LogPerService]** ENABLED|DISABLED *(default: DISABLED)*

    When multiple services are specified (Section 6.3) a separate log file will be created for each if this is enabled. See Section 6.11.3.

80. **[LogPerServiceHostOnly]** ENABLED|DISABLED *(default: DISABLED)*

    When generating a log name do not make the port number part of it. This effectively provides a single log file for all ports provided against a host name (e.g. a standard HTTP service on port 80 and an SSL service on port 443 would have entries in the one file). See Section 6.11.3.

81. **[Monitor]** ENABLED|DISABLED *(default: DISABLED)*

    Allows monitoring via the HTTPDMON utility (Section 23.8. Adds slight request processing overhead.

82. **[OpcomAdmin]** ENABLED|DISABLED *(default: DISABLED)*

    Report to operator log and any enabled operator console (see [OpcomTarget]) server administration directives originating from the Server Administration Menu, for example path map reload, server restart, etc.

83. **[OpcomAuthorization]** ENABLED|DISABLED *(default: DISABLED)*

    Report events related to authentication/authorization. For example username-password validation failures.

84. **[OpcomControl]** ENABLED|DISABLED *(default: DISABLED)*

    Report HTTPD/DO=*directive* control events, both the command-line directive and the server's response.

85. **[OpcomHTTPd]** ENABLED|DISABLED *(default: DISABLED)*

    Report events concerning the server itself. For example, server startup and exit (either normally or with error status).

86. **[OpcomProxyMaint]** ENABLED|DISABLED *(default: DISABLED)*

    Report events related to proxy server cache maintenance. For example, the commencement of file cache reactive and proactive purging, the conclusion of this purge, both with cache device statistics.

87. **[OpcomTarget]** ***string*** *(default: DISABLED)*

    This enables OPCOM messaging and specifies the target for the OPCOM reports. This must be set to a target to enable OPCOM messages, irrespective of the setting of any of the other [Opcom...] directives. These messages are added to SYS$MANAGER:OPERATOR.LOG and displayed at the specified operator's console if enabled (using REPLY/ENABLE=target). The operator log provides a ''permanent'' record of server events. Possible settings include CENTRAL, NETWORK, SECURITY, OPER1 . . . OPER12, etc.

88. **[Port]** ***integer*** *(default: 80)*

    IP port number for server to bind to. For anything other than a command-line server control (Section 5.5.2) this parameter is overridden by anything supplied via the [Service] directive.

89. **[ProxyCache]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables proxy caching on a whole-of-server basis, irrespective of any proxy services that might be configured for caching.

90. **[ProxyCacheFileKBytesMax]** ***integer*** *(default: 256)*

    Maximum size of a cache file in kilobytes before it will not be cached.

91. **[ProxyCacheRoutineHourOfDay]** ***integer*** *(default: 0)*

    Hour of day for *routine* cache purge (00-23).

92. **[ProxyCacheDeviceCheckMinutes]** ***integer*** *(default: 15)*

    Interval in minutes between checking space availablility on cache device. If space is not available a *reactive* purge is initiated.

93. **[ProxyCacheDeviceDirOrg]** FLAT256|64X64 *(default: FLAT256)*

    Organization of directories on the proxy cache device. The first provides a single level structure with a possible 256 directories at the top level and files organized immediately below these. For versions of VMS prior to V7.2 exceeding 256 files per directory, or a total of approximately 65,000 files, incurs a significant performance penalty for some directory operations. The second organization involves two levels of directory, each with a maximum

of 64 directories. This allows for approximately 1,000,000 files before encountering the 256 files per directory issue.

94. **[ProxyCacheDeviceMaxPercent]** ***integer*** *(default: 85)*

    The maximum percentage in use on the cache device before a *reactive* purge is scheduled. If device usage exceeds this limit no more cache files are created.

95. **[ProxyCacheDevicePurgePercent]** ***integer*** *(default: 1)*

    The percentage by which the cache device usage is attempted to be reduced when a *reactive* purge is initiated.

96. **[ProxyCacheNoReloadSeconds]** ***integer*** *(default: 0)*

    Prevents pragma reloads actually retrieving the file from the source host again until the period expires. This is designed to limit concurrent or repeated reloads of files into the cache unecessarily. Thirty seconds is probably an adequate period balancing effect against a user legitimately needing to recache the document.

97. **[ProxyCachePurgeList]** ***string*** *(default: 168,48,24,8,0)*

    A list of comma-separated integers representing the sequence of last accessed period in hours used during a progressive *reactive* purge.

98. **[ProxyCacheReloadList]** ***string*** *(default: 1,2,4,8,12,24,48,96,168)*

    A list of comma-separated integers representing the sequence of age in hours used when determining whether a cache file's contents should be reloaded.

99. **[ProxyForwarded]** BY|DISABLED|FOR|ADDRESS *(default: DISABLED)*

    BY enables the addition of a proxy request header line providing information that the request has been forwarded by another agent. The added header line would look like ''Forwarded: by http://server.name.domain (HTTPd-WASD/n.n.n OpenVMS/AXP Digital-TCPIP SSL)''. If the FOR variant is used the field included the host name (or ADDRESS) the request is being forwarded on behalf of, as in ''Forwarded: by http://server.name.domain (HTTPd-WASD/n.n.n OpenVMS/AXP Digital-TCPIP SSL) for host.name.domain''.

100. **[ProxyHostLookupRetryCount]** ***integer*** *(default: 0)*

     When the server is resolving the name of a remote host the request may timeout due to up-stream DNS server latencies. This parameter allows a number of retries, at five second intervals, to be enabled.

101. **[ProxyReportLog]** ENABLED|DISABLED *(default: DISABLED)*

     Enables or disables the server process log reporting siginificant proxy processing events, such as cache maintenance activity.

102. **[ProxyReportCacheLog]** ENABLED|DISABLED *(default: DISABLED)*

     Enables or disables the server process log reporting of proxy caching activity.

103. **[ProxyServing]** ENABLED|DISABLED *(default: DISABLED)*

     Enables or disables proxy serving on a whole-of-server basis, irrespective of any proxy services that might be configured.

104. **[ProxyXForwardedFor]** ADDRESS|DISABLED|ENABLED|UNKNOWN *(default: DISABLED)*

Enables the addition of a proxy request header line providing the host name on behalf of which the request is being proxied. The added header line would look like ''X-Forwarded-For: host.name.domain''. THE ADDRESS variant provides the IP address, and the UNKNOWN variant substitutes ''unknown'' for the host. This field is degined to be compatible with the *Squid* de facto standard field of the same name. Any request with an existing ''X-Forwarded-For:'' field has the local information appended to the existing as a comm-separated list. The first host in the field should be the original requesting client.

105. **[PutMaxKBytes]** ***integer*** *(default: 250)*

Maximum size of an HTTP POST or PUT method request in Kilobytes.

106. **[PutVersionLimit]** ***integer*** *(default: 3)*

File created using the POST or PUT methods have the specified version limit applied.

107. **[RegEx]** ENABLED|DISABLED *(default: DISABLED)*

Enable regular expression matching. With the possibility of the reserved character ''^'' being used in existing mapping rules regular expression string matching (Chapter 11) is only available after enabling this directive.

108. **[Reject]** ***host/domain name*** *(default: none)*

One or more (comma-separated if on the same line) internet host/domain names, with ''*'' wildcarding for host/subdomain matching, to be explicitly denied access. If DNS lookup is not enabled hosts must be expressed using literal addresses (see [DNSLookup] directive). Also see the [Accept] directive. Reject directives have precedence of Accept directives. The Reject directive may be used multiple times.

Example:

```
[Reject]
*.wasd.dsto.defence.gov.au
131.185.250.*
```

109. **[ReportBasicOnly]** ENABLED|DISABLED *(default: DISABLED)*

Only ever supply basic information in a report (Section 6.9).

110. **[ReportMetaInfo]** ENABLED|DISABLED *(default: DISABLED)*

Includes in detailed reports, as <META> information, the software ID of the server and any relevant VMS file information.

111. **[RequestHistory]** ***integer*** *(default: 0)*

The server can keep a list of the most recent requests accessable from the Server Administration page. This value determines the number kept. Zero disables the facility. Each retained request consumes 256 bytes and adds a small amount of extra processing overhead.

112. **[Scripting]** ENABLED|DISABLED *(default: ENABLED)*

Enables and disables **all** scripting mechanisms. This includes subprocess CGI and CGIplus, DECnet-based OSU and CGI, and SSI directives that DCL subprocesses to provide <–#dcl –>, <–#exec –>, etc.

113. **[SearchScript]** ***path*** *(no default)*

Specifies the **URL-format path** to the default query-string keyword search script. This path can subsequently be remapped during request processing.

Example:

```
[SearchScript] /ht_root/script/query
```

114. **[SearchScriptExclude]** ***list*** *(no default)*

Provides a list of file types that are excluded from an implied keyword search. This is useful for client-side (browser-side) active processing that may require a query string to pass information. This query string would normally be detected by the server and if not in a format to be meaningful to itself is then considered as an implied (HTML <ISINDEX>) keyword search, with the approriate script being activiated.

Example:

```
[SearchScriptExclude] .HTA,.HTL
```

115. **[ServerAdmin]** ***string*** *(no default)*

Specifies the contact email address for server administration issues. Included as a ''mailto:'' link in the server signature if [ServerSignature] is set to *email*.

116. **[ServerAdminBodyTag]** ***string*** *(default: <BODY>)*

Specifies the HTML <BODY> tag for server administration and administration report pages. This allows some measure of control over the ''look-and-feel'' of page and link colour, etc.. for the administrator.

117. **[ServerReportBodyTag]** ***string*** *(default: <BODY>)*

Specifies the HTML <BODY> tag for server error and other report pages. This allows some measure of site ''look-and-feel'' in page colour, background, etc. to be maintained.

118. **[ServerSignature]** ENABLED|EMAIL|DISABLED *(default: DISABLED)*

The server signature is a short identifying string added to server generated error and other report pages. It includes the server software name and version, along with the host name and port of the service. Setting this to *email* makes the host name a *mailto:* link containing the address specified by the [ServerAdmin] directive.

119. **[Service]** ***string*** *(no default)*

This parameter allows SSL, multi-homed hosts and multiple port serving to be specified, see Section 17.3 and Section 6.3.

120. **[ServiceNotFoundURL]** ***string*** *(no default)*

Provides a default path for reporting a virtual host does not exist, see Unknown Virtual Server.

121. **[SSI]** ENABLED|DISABLED *(default: DISABLED)*

Enables or disables Server Side Includes (HTML pre-processing).

122. **[SSIaccesses]** ENABLED|DISABLED *(default: DISABLED)*

Enables or disables Server Side Includes (HTML pre-processing) file access counter.

123. **[SSIexec]** ENABLED|DISABLED *(default: DISABLED)*

Enables or disables Server Side Includes (HTML pre-processing) DCL execution functionality.

124. **[SSIsizeMax]** ***integer*** *(default: 0 (128kB))*

SSI source files a completely read into memory before processing. This allows the maximum size to be expanded beyond the default.

125. **[StreamLF]** ***integer*** *(default: 0 (disabled))*

Enables or disables automatic conversion of VARIABLE record format documents (files) to STREAM-LF, which are much more efficient with this server. The integer is the maximum size of a file in kilobytes that the server will attempt to convert. Zero disables any conversions. See File Record Format.

126. **[StreamLFpaths]** ***string*** *(no default)*

(Retired in v5.3, mapping SET rule provides this now, see Section 13.4.5).

127. **[TimeoutInput]** ***hh:mm:ss*** *(default: 00:01:00)*

Period allowing a connection request to be in progress without submitting a complete request header before terminating it.

128. **[TimeoutKeepAlive]** ***hh:mm:ss*** *(default: 00:00:05)*

Period allowing a ''Keep-Alive:'' request connection is maintained after the conclusion of a request. *Keep-Alive* improves the overall performance of the server by reducing the number of discrete TCP/IP connections that need to be estabiliched.

129. **[TimeoutNoProgress]** ***hh:mm:ss*** *(default: 00:02:00)*

Period allowing request output to continue without any increase in the number of bytes transfered. This directive is targeted at identifying and eliminating requests that have stalled.

130. **[TimeoutOutput]** ***hh:mm:ss*** *(default: 00:10:00)*

Period allowing a request to be output before terminating it. This directive sets an absolute maximum time a request can continue to receive output.

131. **[Track]** ENABLED|DISABLED *(default: DISABLED)*

Enables session (user) tracking.

132. **[TrackMultiSession]** ENABLED|DISABLED *(default: DISABLED)*

By default the tracking cookie is discarded by the browser when it is closed. This setting directs the browser to keep it between sessions.

133. **[TrackDomain]** ***file.suffix*** *(no default)*

User tracking applies only to the originating (virtual) server by default. This directive allows it to applied to all a particular domain's sites. Top-level sites (e.g. those in ''.com'', ''.edu'' etc.) would specify this as something like *organization.domain* (i.e. two dots), while other would use *organization.group.domain* (i.e. three dots).

134. **[Welcome]** ***file.suffix*** *(no default)*

Specifies the names and order in which a directory is checked for home page files. If no home page is found a directory listing is generated.

```
[Welcome]
index.html
index.htm
home.html
home.htm
```

Dynamic home pages (script or interpreter engine driven, e.g. Perl, PHP) may be deployed using a combination of the [Welcome] and [DclScriptRunTime] directives.

```
[Welcome]
index.html
index.htm
index.php
index.pl

[DclScriptRunTime]
.PHP $CGI-BIN:[000000]PHPWASD.EXE
.PL $CGI-BIN:[000000]PERLRTE
```

Chapter 9
Service Configuration

Note

The service configuration file is optional. If the HTTPD$SERVICE logical is not defined or the file does not exist service configuration is made using the HTTPD$CONFIG [Service] directives. For simple sites, those containing one or two services, the use of a separate service configuration file is probably not warranted. Once the number begins to grow this file offers a specific management interface for those services.

Precedence of service specifications:

1. /SERVICE= command line qualifier
2. HTTPD$SERVICE configuration file (if logical defined and file exists)
3. HTTPD$CONFIG [Service] directive

There is no design limit to the number of services that WASD can support. It can listen on any number of IP ports and for any number of virtual services for any given port.

Specific Services

In common with other configuration files, directives associated with a specific virtual services are introduced using a double-bracket delimited host specification (Section 6.3). When configuring a service the following three components specify its essential characteristics.

- **scheme -** HTTP scheme (sometimes refered to as *protocol*). If *http:* (or omitted) it is a standard HTTP service. If *https:* an SSL service is configured.
- **host -** Host name or dotted-decimal address. If omitted, or specified as an asterisk (‘‘*’’), defaults to the system’s IP host name.
- **port -** IP port the service is offered on. If omitted it defaults to 80 for an *http:* service, and to 443 for an *https:* (SSL) service.

Generic Services

A *generic* service is one that specifies a scheme and/or port but no specific host name. This is useful in a cluster where multiple systems all provide a basic service (e.g. a port 80 service). If the host name is omitted or specified as an asterisk the service substitutes the system's IP host name. See Section 9.5.

SSL Services

Multiple virtual SSL services (https:) sharing the same certificate can essentially be configured against any host name (unique IP address or alias) and/or port in the same way as standard services (http:). Services requiring unique certificates can only be configured for the same port number against individual and unique IP addresses (i.e. not against aliases). This is not a WASD restriction, it applies to all servers for significant SSL technical reasons (Chapter 17).

For example, unique certificates for https://www.company1.com:443/ and https://www.company2.com:443/ can be configured only if COMPANY1 and COMPANY2 have unique IP addresses. If COMPANY2 is an alias for COMPANY1 they must share the same certificate. During startup service configuration the server checks for such conditions and issues a warning about ''sharing'' the service with the first configured.

Administration Services

When multiple instances are configured Server Administration page access, in common with all request processing, is automatically shared between those instances. There are occasions when consistent access to a single instance is desirable. The [ServiceAdmin] directive indicates that the service port number should be used as a base port and all instances create their own service with unique port for access to that instance alone. The first instance to create an *administration service* uses the specified port, or the next successive if it's already in use, the next instance will use the next available port number, and so on. A high port number should be specified. The Server Administration page lists these services for all server instances in the cluster. This port configuration is not intended for general request activity, although with appropriate mapping and other configuration there is nothing specifically precluding it's use (remembering that the actual port in use by any particular instance may vary across restarts). In all other respects the services can (and should) be mapped, authorized and otherwise configured as any other. See Section 9.5.

9.1 IPv4 and IPv6

Both IP version 4 and 6 are concurrently supported by WASD. All networking functionality, service creation, SSL, proxy HTTP, proxy FTP and RFC1413 authorization is IPv6 enabled. If system TCP/IP services do not support IPv6 the expected error would be

```
%SYSTEM-F-PROTOCOL, network protocol error
```

during any attempted IPv6 service creation. Of course IPv4 service creation would continue as usual.

Server configuration handles the standard dotted-decimal addresses of IPv4, as well as ‘‘normal’’ and ‘‘compressed’’ forms of standard IPv6 literal addresses, and a (somewhat) standard variation of these that substitutes hyphens for the colons in these addresses to allow the colon-delimited port component of a ‘‘URL’’ to be resolved.

IPv6 Literal Addresses

Normal	**Compressed**
1070:0:0:0:0:800:200C:417B	1070::800:200C:417B
0:0:0:0:0:0:13.1.68.3	::13.1.68.3
0:0:0:0:0:FFFF:129.144.52.38	::FFFF:129.144.52.38
hyphen-variants	
1070-0-0-0-0-800-200C-417B	1070–800-200C-417B
0-0-0-0-0-0-13.1.68.3	–13.1.68.3
0-0-0-0-0-FFFF-129.144.52.38	–FFFF-129.144.52.38

In common with all virtual services, if a connection can be established with the system and service port the HTTPd can respond to that request. If a service needs to be bound to a specific IP address then that can be specified using the [ServiceBind] directive using any of the literal address formats described above.

9.2 Service Directives

Where a service directive has an equivalent configuration directive (e.g. error report path) the service directive takes precedence. This allows specific virtual services to selectively override the generic configuration.

Service Directives

[[virtual-service]]	scheme://host:port
[ServiceAdmin]	an *instance* Server Administration page service
[ServiceBind]	if different to host’s
[ServiceBodyTag]	<BODY> tag for server reports., etc
[ServiceClientSSLcipherList]	proxy SSL connect ciphers
[ServiceClientSSLverifyCA]	verify CA of proxied requests
[ServiceClientSSLverifyCAfile]	location of proxy CA file
[ServiceClientSSLversion]	proxy SSL version to use
[ServiceErrorReportPath]	path to script, SSI or ‘‘flat’’ error document

[ServiceNoLog]	suppress logging
[ServiceNoTrack]	suppress user tracking
[ServiceProxy]	proxy service
[ServiceProxyAuth]	require proxy authorization
[ServiceProxyCache]	proxy caching
[ServiceProxyChain]	chained proxy service host
[ServiceProxySSL]	proxy SSL enable
[ServiceProxyTrack]	user track proxy access
[ServiceProxySSL]	provide proxy of SSL (connect:)
[ServiceSSLcert]	SSL service certificate
[ServiceSSLcipherList]	list of accepted SSL ciphers
[ServiceSSLkey]	SSL service private key
[ServiceSSLverifyPeer]	access only using verified peer certificate
[ServiceSSLverifyPeerCAfile]	location of CA file
[ServiceSSLversion]	SSL version to use

9.3 Directive Detail

Some of these directives control the behaviour of proxy services. Refer to Chapter 16 for further detail. Other directive are Secure Sockets Layer (SSL) specific. This is an optional package described in Chapter 17.

1. **[[virtual-service]]** *(default: none)*

   Specifies the scheme, host name (or asterisk) and port of a service.

2. **[ServiceAdmin]** ENABLED|DISABLED *(default: DISABLED)*

   Marks the port as *administration* service (Administration Services).

3. **[ServiceBind]** ***literal address*** *(default: none)*

   If the system has a multi-homed network interface this binds the service to the specific IP address and not to INADDR_ANY. Generally this will not be necessary. The literal address may be in IPv4 dotted-decimal or IPv6 normal or compressed hexdecimal.

4. **[ServiceBodyTag]** ***string*** *(default: <BODY>)*

   Specifies the HTML <BODY> tag for server error and other report pages. This allows some measure of site ‘‘look-and-feel’’ in page colour, background, etc. to be maintained.

5. **[ServiceClientSSL]** ENABLED|DISABLED *(default: DISABLED)*

   Enables a proxy service to *originate* HTTP-over-SSL requests. This is different to the CONNECT service enabled using [ServiceProxySSL]. It allows requests to be gatewayed between standard HTTP and Secure Sockets Layer (see Section 16.5).

6. **[ServiceClientSSLcipherList]** ***string*** *(default: none)*

   A comma-separated list of SSL ciphers to be used by the gateway to connect to SSL services. The use of this parameter might allow the selection of stronger ciphers to be forced to be used or the connection not allowed to procede.

   Note

   These *ServiceClientSSL..* directives are used to control behaviour when outgoing SSL connections are established (as with HTTP-to-SSL gatewaying). This should not be confused with verification of client certificates, which is better refered to as peer verification. See [ServiceSSLverifyPeer] and [ServiceSSLverifyPeerCAfile] directives.

7. **[ServiceClientSSLverifyCA]** ENABLED|DISABLED *(default: DISABLED)*

   Unless this directive is enabled the Certificate Authority (CA) used to issue the service's certificate is not verified. Requires that a CA file be provided. See note in [Service-ClientSSLcipherList] above.

8. **[ServiceClientSSLverifyCAfile]** ***string*** *(default: none)*

   Specifies the location of the collection of Certificate Authority (CA) certificates used to verify the connected-to server's certificate (VMS file specification). See note in [ServiceClientSSLcipherList] above.

9. **[ServiceClientSSLversion]** ***string*** *(default: SSLV2/V3)*

   The abbreviation for the SSL protocol version to be used to connect to the SSL service. See note in [ServiceClientSSLcipherList] above.

10. **[ServiceErrorReportPath]** ***string*** *(default: none)*

    Specifies the **URL-format path** to an optional, error reporting SSI document or script (Section 6.9). This path can subsequently be remapped during request processing.

11. **[ServiceNoLog]** ENABLED|DISABLED *(default: DISABLED)*

    When request logging is enabled then by default all services are logged. This directive allows logging to be suppressed for this service.

12. **[ServiceNoTrack]** ENABLED|DISABLED *(default: DISABLED)*

    When use tracking is enabled then by default all requests on non-proxy services are tracked. This directive allows tracking to be suppressed for this service.

13. **[ServiceProxy]** ENABLED|DISABLED *(default: DISABLED)*

    Enables and disables proxy request processing for this service (Chapter 16).

14. **[ServiceProxyAuth]** ***NONE***|AUTH|LOCAL *(default: none)*

    Makes a proxy service require authorization before a client is allowed access via it (Section 16.1.4). NONE disables authorization. PROXY enables HTTP proxy authorization. LOCAL enables standard server authorization.

15. **[ServiceProxyCache]** ENABLED|DISABLED *(default: DISABLED)*

    Enables and disables proxy caching for a proxy service.

16. **[ServiceProxyChain]** ***string*** *(default: none)*

    Specifies the next proxy host if chained.

17. **[ServiceProxyTrack]** ENABLED|DISABLED *(default: DISABLED)*

    When user tracking is enabled only non-proxy services have it applied by default. This directive allows proxy service usage tracking to be enabled.

18. **[ServiceProxySSL]** ENABLED|DISABLED *(default: DISABLED)*

    Specifies the service as providing proxying of SSL requests. This is sometimes refered as a ''CONNECT'' service. This proxies ''https:'' requests directly and is different to the HTTP-to-SSL proying described in Section 16.3 and enabled using [ServiceProxyHttpSSL].

19. **[ServiceSSLcert]** ***string*** *(default: none)*

    Specifies the location of the SSL certificates (VMS file specification) (Section 17.3).

20. **[ServiceSSLcipherList]** ***string*** *(default: none)*

    A comma-separated list of SSL ciphers allowed to be used by clients to connect to SSL services. The use of this parameter might allow the selection of stronger ciphers to be forced to be used or the connection not allowed to procede.

21. **[ServiceSSLkey]** ***string*** *(default: none)*

    Specifies the location of the SSL private key (VMS file specification).

22. **[ServiceSSLverifyPeer]** ENABLED|DISABLED *(default: DISABLED)*

    To access this service a client must provide a verified CA client certificate (Section 17.3.7).

23. **[ServiceSSLverifyPeerCAfile]** ***string*** *(default: none)*

    Specifies the location of the collection of Certificate Authority (CA) certificates used to verify a peer certificate (VMS file specification, Section 17.3.7).

24. **[ServiceSSLversion]** ***string*** *(default: SSLV2/V3)*

    The abbreviation for the SSL protocol version allowed to be used to connect to an SSL service. Using the directive a service may select prefered (read stronger) protocols.

9.4 Administration

A service configuration file can be maintained using a simple text editor and HTTPD$SERVICE.

Alternatively the Server Administration page may be used (Chapter 18). When using this interface for the first time ensure the HTTPD$SERVICE logical is correctly defined. If the file did not exist at server startup any services will have been created from the HTTPD$CONFIG [Service] directive. These will be displayed as the existing services and will be saved to the configuration file the first time it is saved.

The [IncludeFile] is a directive common to all WASD configuration, allowing a separate file to be included as a part of the current configuration (Section 6.12).

Not all configuration directives may be shown depending on the type of service. For instance, unless a service is configured to provide proxy, only the [ServiceProxy] directive is displayed. To fully configure such a service enable it as proxy, save the file, then reload it. The additional directives will now be available.

There is always one empty service displayed each time the configuration menu is generated. This information may be changed appropriately and then saved to add new services to the configuration (of course, these will not be available until the server is restarted). To configure multiple new services add one at a time, saving each and reloading the file to provide a new blank service.

9.5 Examples

1. The following example shows three services being configured. The first is standard HTTP on the default (and well-known) port 80. The second is a proxy service on port 8080. This service provides both standard HTTP (with response caching enabled), SSL (connect:) access and proxy authorization required. The third service is SSL, with a host-specific certificate and key.

```
[[http://alpha.domain.name:80]]

[[http://alpha.domain.name:8080]]
[ServiceProxy]  enabled
[ServiceProxyAuth]  PROXY
[ServiceProxyCache]  enabled
[ServiceProxySSL]  enabled

[[https://alpha.domain.name:443]]
[ServiceSSLcert] ht_root:[local]alpha.pem
```

2. This example shows a generic service service being configured on the well-known port 80.

```
[[http://*:80]]
```

   If a cluster of four systems, ALPHA, BETA, GAMMA and DELTA all use this configuration each will have a service accessable via the following four URLs.

```
http://alpha.domain.name/
http://beta.domain.name/
http://gamma.domain.name/
http://delta.domain.name/
```

3. The following example show two services configured against specific IP addresses. The first is an IPv4 and the second a compressed IPv6.

```
[[http://alpha4.domain.name:80]]
[ServiceBind]  168.192.0.3

[[https://alpha6.domain.name:80]]
[ServiceBind]  fe80::200:f8ff:fe24:1a22
```

4. An *administration port* is a special configuration used to support the Server Administration page (Chapter 18) when multiple per-node instances are configured (Section 6.2). See description above.

```
[[https://alpha.domain.name:44443]]
[ServiceAdmin]  enabled
[ServiceSSLcert] ht_root:[local]alpha.pem
[ServiceSSLkey] ht_root:[local]alpha.pem
```

Chapter 10
Message Configuration

Message configuration is provided for two purposes.

1. Some sites would prefer to customize or extend the basic information provided to clients when an error or other event occurs.
2. Sites that do not use English as a first language may wish to provide some or all of the defined messages using a prefered language.

Not all messages provided by the WASD server are customizable, only those generated for non-administrative content. As the WASD server can also report using information derived from the standard VMS message service (via *sys$getmsg( )*) it is assumed a language-local implementation of this is in use as well. Unfortunately for the non-first-language-English Web and system administrators, the menus and messages used for administration purposes, etc., are still only in English. The intent of this facility is to provide non-administration clients only with a more familiar language environment.

Also note that the message database only applies to messages generated by the server, not to any generated by scripts, etc.

10.1 Behaviour

When an error, or other message or string, needs to be provided for the client the message database is accesssed using the following algorithm.

1. If the client request has specified a list of prefered languages using the ''Accept-Language:'' HTTP header field the message database is checked for support of that/those languages. If one is found then that language is used to access the message.
2. If none is found, or the client has not specified a prefered language, the client host address is checked against any list of hosts/domains provided against the language (see below). If a match occurs the specified language is used.
3. If neither of the above results in a message language the base language is used (the highest numbered language). This **must** have a complete set of messages or the server will not start!

10.2 Message File Format

By default, the system-table logical name HTTPD$MSG locates a common message file, unless an individual message file is specified using a job-table logical name. Simple editing of the message file changes the messages (after a server restart, of course). Comment lines may be included by prefixing them with the hash character (‘‘#’’), and lines continued by ensuring the last character is a backslash (‘‘\’’). The server will concurrently support an additional 3 languages to the base English (although this can be increased by recompilation :^)

Note

Care must be taken with the message file or the server may refuse to start!
Worst-case; the HTTPD$MSG.CONF message file may be copied from [EXAMPLE].

As illustrated below the message file comprises a series of sections. Directives enclosed by square-brackets provide information to the message loader.

```
# this is a comment

[version]   8.0
[language]  1  es,es-ES,es-AR  hosts=*.es,*.ar
[language]  2  de,de-*         hosts=*.de,*.at
[language]  3  en

[general]

en 01 Sanity check failure.
de 01 (the same message but in German, which I cannot speak :^)
es 01 (the same message but in Spanish, which I also cannot speak :^)
en 02 String overflow.
de 02 (the same message but in German, which I cannot speak :^)
es 01 (the same message but in Spanish, \
which I still cannot speak :^) \
this has also been continued over two additional lines!
en 03 Heap allocation failed.
en 04 calloc() failed
en 05 Request calloc() failed.
en 06 Server too busy.
en 07 Server access denied.
en 08 Facility is disabled.
en 09 Wildcard not permitted.
en 10 Directory layout problem.

[next-section, etc.]
```

The square-bracketed section headings have the following functions.

- **[version] -** Ensures the correct database version is available for the server version attempting to use it. The message file always needs checking for this version number being changed at server updates, although the version may remain fixed at a previous server version number if there have been no changes to the message database during subsequent server versions. This must be the first directive in the file.
- **[language] -** Creates space for assigning the new language’s messages. The number specifies an order within the languages, each must be different, but only the lowest and highest (prefered and base respectively) have operational significance. The highest number should always be English to provide a fall-back message. A short string provides an identifier for the language. This identifier should be the same as the identifying string

in the browser request ''Accept-Language:'' header field (e.g. ''en'', ''se'', ''de'', ''fr'', etc.) Multiple, comma-separated languages may be specified. The first is the primary language of that list and messages must be specified using that. The subsequent languages are equivalents that might be specified by the client. A wildcard may be used to match all possibilities (e.g. ''de,de-*'', ''es,es-*''). Following the language identifier is an optional host/domain list. Multiple hosts/domains may be specified by separating each with a comma. The specifications may contain wildcards. All the [language] directives should be grouped at the start of the file immediately following the [version] directive. A character set may be associated with a particular language by specifying a *charset=* following the language string (e.g. ''ru charset=koi8-r''). Setting the language's ordering number to zero disables the language completely. All messages associated with it will then be ignored.

- **[group-name] -** The messages are divided into groupings to make them easier to manage. Each group begins with the group name directive.
- **en 01 message -** Each message in a group is assigned using using this format. The string identifying the language, then the message number (the leading zero just improves the format, strictly it is not required), then the actual message itself. The message can be of arbitrary length. Long messages may be continued on following lines using the ''\'' continuation character.

The base language (the highest numbered, which should always be English) must have precisely the right number of messages required by the server, too few or too many and the server will not start! **Additional languages do not have to reassign every message!** The base language will supply any not assigned. A message number of zero is disabled and completely ignored.

If messages contain HTML tags that markup must not interfere with the general HTML page it is used within.

Some messages are a composite of multiple strings each of which is used on a different part of the one page (e.g. for the [upd] edit-page). Each of the strings is delimited by the vertical bar ''|''. Care must be taken when customizing these strings that the overall number stays the same and that the length of each does not become excessive. Although it will not disrupt the server it may significantly disrupt the page layout.

All message numbers must be included. To provide an empty string for any one message (not recommended) provide the line with nothing following the message number.

10.3 Supplied Message Files

Any non-English message files that are provided to the author will be included for general use (please take the time to support this endeavour) in the HT_ROOT:[EXAMPLE] directory.

online hypertext link

Note that message files can become out-of-date as server versions change, requiring modifications to the message database. Check the version information and/or comments at the top of candidate message files, however even slightly dated files may serve as a good starting point for a locale-specific message base.

Chapter 11
String Matching

Matching of strings is a pervasive and important function within the server. Two types are supported; wildcard and regular expression. Wildcard matching is generally much less expensive (in CPU cycles and time) than regular expression matching and so should always be used unless the match explicitly requires otherwise. WASD attempts to improve the efficiency of both by performing a preliminary pass to make simple matches and eliminate obvious mismatches using a very low-cost comparison. This either matches or doesn't, or encounters a pattern matching meta-character which causes it to undertake full pattern matching.

To assist with the refinement of string matching patterns the Server Administration facility (Chapter 18) has a report item named ''Match''. 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 (Section 11.4) allows resultant strings to be assessed.

To determine what string match processing is occuring during request processing in the running server use the *match* item available from the Server Administration WATCH Report (Chapter 19).

11.1 Wildcard Patterns

Wildcard patterns are simple, low-cost mechanisms for matching a string to a template. They are designed to be used in path and authorization mapping to compare a request path to the root (left-hand side) or a template expression.

Wildcard Operators

Expression	Purpose
*	Match zero or more characters (non-greedy)
**	Match zero or more characters (greedy)

Expression	Purpose
%	Match any one character

Wildcard matching uses the '*' and '%' symbols to match any zero or more, or any one character respectively. The '*' wildcard can either be greedy or non-greedy depending on the context (and for historical reasons). It can also be forced to be greedy by using two consecutive ('**'). By default it is not greedy when matching request paths for mapping or authentication, and is greedy at other times (matching strings within conditional testing, etc.)

Greedy and Non-Greedy

Non-greedy matching attempts to match an asterisk wildcard up until the first character that is not the same as the character immediately following the wildcard. It matches a minimum number of characters before failing. Greedy matching attempts to match all characters up until the first string that does not match what follows the asterisk.

To illustrate; using the following string

```
non-greedy character matching compared to greedy character matching
```

the following non-greedy pattern

```
*non-greedy character*matching
```

does not match but the following greedy pattern

```
*non-greedy character**matching
```

does match. The non-greedy one failed as soon as it encountered the space following the first ''matching'' string, while the greedy pattern continued to match eventually encountering a string matching the string following the greedy wildcard.

11.2 Regular Expressions

Regular expression matching is case insensitive (in line with other WASD behaviour) and uses the Posix EGREP pattern syntax and capabilities. Regular expression matching offers significant but relatively expensive functionality. One of those expenses is expression compilation. WASD attempts to eliminate this by pre-compiling expressions during server startup whenever feasable. Regular expression matching must be enabled using the [RegEx] HTTPD$CONFIG directive and are then differentiated from wildcard patterns by using a leading ''^'' character.

A detailed tutorial on regular expression capabilities and usage is well beyond the scope of this document. Many such hard-copy and on-line documents are available. This summary is only to serve as a quick mnemonic. WASD regular expressions support the following set of operators.

Operator Overview

Description	Usage
Match-self Operator	Ordinary characters.
Match-any-character Operator	.
Concatenation Operator	Juxtaposition.
Repetition Operators	* + ? {}
Alternation Operator	|
List Operators	[...] [^...]
Grouping Operators	(...)
Back-reference Operator	\digit
Anchoring Operators	^ $
Backslash Operator	Escape meta-character; i.e. \ ^ . $ | [(

The following operators are used to match one, or in conjunction with the repetition operators more, characters of the target string. These single and leading characters are reserved meta-characters and must be escaped using a leading backslash (''\'') if required as a literal character in the matching pattern.

Matching Operators

Expression	Purpose
^	Match the beginning of the line
.	Match any character
$	Match the end of the line
|	Alternation (or)
[abc]	Match only a, b or c
[^abc]	Match anything except a, b and c
[a-z0-9]	Match any character in the range a to z or 0 to 9

Repetition operators control the extent, or number, of whatever the matching operators match. These are also reserved meta-characters and must be escaped using a leading backslash if required as a literal character.

Repetition Operators

Expression	Function
*	Match 0 or more times

Expression	Function
+	Match 1 or more times
?	Match 1 or zero times
{n}	Match exactly n times
{n,}	Match at least n times
{n,m}	Match at least n but not more than m times

11.3 Examples

The following provides a series of examples as they might occur in use for server configuration.

1. Equivalent functionality using wildcard and regular expression patterns. Note that ''Mozilla'' must be at the start of the string, with the regular expression using the start-of-string anchor resulting in two consecutive ''^''s, one indicating to WASD a regular expression, the other being part of the expression itself.

```
if (user-agent:Mozilla*Gecko*)
if (user-agent:^^Mozilla.*Gecko)
```

2. This shows path matching using equivalent wildcard and regular expression matching. Note the requirement to use the regular expression *grouping* parentheses to provide the substitution elements, something provided implicitly with wildcard matching.

```
map /*/-/* /ht_root/runtime/*/*
map ^/(.+)/-/(.+) /ht_root/runtime/*/*
```

3. This rather contrived regular expression example has no equivalent capability available with wildcard matching. It forbids the use of any path that contains any character other than alpha-numerics, the hyphen, underscore, period and forward-slash.

```
pass ^[^-_./a-z0-9]+ "403 Forbidden character in path!"
```

11.4 Expression Substitution

Expression substitution is available during path mapping (Chapter 13). Both wildcard (implicitly) and regular expressions (using *grouping* operators) note the offsets of matched portions of the strings. These are then used for wildcard and *specified* wildcard substitution where result strings provide for this (e.g. mapping 'pass' and 'redirect' rules). A maximum of nine such wildcard substitutions are supported (one other, the zeroeth, is the full match).

Wildcard Substitution

With wildcard matching each asterisk wildcard contained in the pattern (*template* string) has matching characters in the *target* string noted and stored. Note that for the percentage (single character) wildcard no such storage is provided. These characters are available for substitution using corresponding wildcards present in the *result* string. For instance, the target string

```
this is an example target string
```

would be matched by the pattern string

```
* is an example target *
```

as containing two matching wildcard strings

```
this
string
```

which could be substituted using the result string

```
* is an example result *
```

producing the resultant string

```
this is an example result string
```

Regular Expression Substitution

With regular expression matching the groups of matching characters must be explicitly specified using the *grouping* parenthesis operator. Hence with regular expression matching it is possible to match many characters from the target string without retaining them for later substitution. Only if that match is designated as a subsitution source do the matching characters become available for substituion via any result string. Using two possible target strings as an example

```
this is an example target string
this is a contrived target string
```

would both be matched by the regular expression

```
^^([a-z]*) is [a-z ]* target ([a-z]*)$
```

which though it contains three regular expressions in the pattern, only two have the grouping parentheses, and so make their matching string available for substitution

```
this
string
```

which could be substituted using the result string

```
* is the final result *
```

producing the resultant string

```
this is the final result string
```

Specified Substitution

By default the strings matched by wildcard or grouping operators are substituted in the same order in which they are matched. This order may be changed by specifying which wildcard string should be substituted where. Not all matched (and stored) strings need to be substituted. Some may be omitted and the contents effectively ignored.

The specified substitution syntax is a result wildcard followed by a single-apostrophe (') and a single digit from zero to nine (0 . . . 9). The zeroeth element is the full matching string. Element one is the first matching part of the expression, on through to the last. Specifying an element that had no matching string substitutes an empty string (i.e. nothing is added). Using the same target string as in the previous previous example

```
this is an example target string
```

and matched by the wildcard pattern string

```
* is an example target *
```

when substituted by the result string

```
*'2 is an example result
```

would produce the resultant string

```
string is an example result
```

with the string represented by the first wildcard effectively being discarded.

Chapter 12
Conditional Configuration

Mapping (HTTPD$MAP) and authorization (HTTPD$AUTH) rules may be conditionally applied depending on request, server or other charactersistics. These include

server host name, port
client IP address and host name
browser-accepted content-types, character sets, languages, encodings
browser identification string
scheme (‘‘http:’’ or ‘‘https:’’, i.e. is it a secure request?)
HTTP method (GET, POST, etc.)
request path, query string, cookie data, refering page
virtual host:port specified in request header
system information (hardware, Alpha/VAX, node name, VMS version, etc.)
local time
random number generation

Conditionals may be nested up to a maximum depth of eight, are not case sensitive and generally match via string comparison, although some tests are performed as boolean operations, by converting the conditional parameter to a number before comparison, and IP address parameters will accept a network mask as well as a string pattern.

String Matching

The basis of much conditional decision making is string pattern matching. Both wildcard and regular expression based pattern matching is available (Chapter 11). Wildcard matching in conditional tests is *greedy*. Regular expression matching, in common with usage throughout WASD, is differentiated from wildcard patterns using a leading ‘‘^’’ character.

12.1 Conditional Syntax

Conditional expressions and processing flow structures may be used in the following formats. Conditional and rule text may be indented for clarifying structure.

```
if (condition) then apply rest of line

if (condition)
   then apply one
   or more rules
   up until the corresponding  . . .
endif

if (condition)
   then apply one
   or more rules
else
   apply one or more other rules
   up until the corresponding  . . .
endif

if (condition)
   then apply one
   or more rules
elif (condition)
   apply one or more other rules
   in a sort or case statement
else
   a possible default rule or rules
   up until the delimiting
endif
```

Logical operators are also supported, in conjunction with precedence ordering parentheses, allowing moderately complex compound expressions to be applied in conditionals.

! logical negation
&& logical AND
|| logical OR

There are two more conditional structures that allow previous decisions to be reused. These are *unif* and the *ifif*. The first unconditionally includes rules regardless of the current state of execution. The second resumes execution only if the previous *if* or *elif* expression was true. The *else* statement may also be used after an *unif* to continue only if the previous expression was false. The purpose of these constructs are to allow a single decision statement to include both conditional and unconditional rules.

```
if (condition)
   then apply one
   or more rules
unif
   apply this block of rules
   unconditionally
ifif
   applied only if the original
   if expression was evaulated as true
unif
   apply another block of rules
   unconditionally
else
   and this block of rules
   only if the original if was false
endif
```

CAUTION

Conditional syntax is checked at rule load time (either server startup or reload). Basic errors such as unknown keywords and unbalanced parentheses or structure statements will be detected and reported to the corresponding Admin Menu report and to the server process log. Unless these reports are checked after modifying rule sets syntax errors may result in unexpected mappings or access. Although the server cannot determine the correct intent of an otherwise syntactically correct conditional, if it encounters an unexpected but detectable condition during processing it aborts the request, supplying an appropriate error message.

12.2 Conditional Keywords

The following keywords provide a match between the corresponding request or other value and a string immediately following the delimiting colon. White space or other reserved characters may not be included unless preceded by a backslash. The actual value being used in the conditional matching may be observed using the mapping item of the WATCH facility (Chapter 19).

Conditional Keywords

Keyword	Description
accept:	Browser-accepted content types as listed in the ''Accept:'' request header field. Same string as provided in CGI variable HTTP_ACCEPT.
accept-charset:	Browser-accepted character sets as listed in the ''Accept-Charset:'' request header field. CGI variable HTTP_ACCEPT_CHARSET.
accept-encoding:	Browser-accepted content encoding as listed in the ''Accept-Encoding:'' request header field. CGI variable HTTP_ACCEPT_ENCODING.
accept-language:	Browser language preferences as listed in the ''Accept-Language:'' request header field. CGI variable HTTP_ACCEPT_LANGUAGE.

Keyword	Description
authorization:	The raw authorization string from the request header, if any supplied. This could be simply used to test whether it has been supplied or not.
callout:	Simple boolean value. If a script callout is in progress (see ''Scripting Overview, CGI Callouts''.) it is true, otherwise false.
client_connect_gt:	An integer representing the current network connections (those currently being processed plus those currently being ''kept alive'') for the particular client represented by the current request. If greater than this value returns true, otherwise false. See Section 6.5.
cluster_member:	If the supplied node name is (perhaps currently) a member of the cluster (if any) the server may be executing on.
command_line:	The command line qualifiers and parameters used when the server image was activated.
cookie:	Raw cookie data as the text string provided in ''Cookie:'' request header field. CGI variable HTTP_COOKIE.
decnet:	Whether DECnet is active on the system and which version is available. This value will be 0 if not active, 4 if PhaseIV or 5 is PhaseV.
document_root:	The DOCUMENT_ROOT CGI variable SET using the *map=root=<string>* mapping rule.
forwarded:	Proxy/gateway host(s) request forwarded by, as specified in request header field ''Forwarded:''. CGI variable HTTP_FORWARDED.
host:	The host (and optionally port) specified in request header ''Host:'' field. This is used by all modern browsers to provide virtual host information to the server. CGI variable HTTP_HOST.
jpi_username:	The account username the server is executing as.
mapped_path:	The path resulting from mapping (phase 2 if script path involved) from which the path-translated is derived.
notepad:	Information (strings) stored using the SET *notepad=* mapping rule. See Notepad: Keyword.
ods:	Specified as 2 or 5 (Extended File System), or as SRI file name encoding (MultiNet NFS and others) PWK encoding (PATHWORKS 4/5), ADS encoding (Advanced Server / PATHWORKS 6), SMB encoding (Samba - same as ADS).
pass:	A numeric value, 1 or 2, representing the first or second pass (if a script component was parsed) through the path mapping rules. Will be zero at other times.
path-info:	Path specified in the request line. CGI variable PATH_INFO.
path-translated:	VMS translation of path-info. Available after rule mapping (i.e. during authorization rule processing).

Keyword	Description
query-string:	Query string specified in request line. Same information as provided in CGI variable QUERY_STRING.
rand:	Value from a random number generator. See Rand: Keyword.
redirected:	If a request has been internally redirected (Section 13.4.2) this conditional will be non-zero. Can be used as a boolean or with a digit specified.
referer:	URL of refering page as provided in ‘‘Referer:’’ request header field. CGI variable HTTP_REFERER.
regex:	Simple boolean value. If configuration directive [RegEx] is enabled (and hence regular expression string matching, Chapter 11) this will be true.
remote-addr:	Client IP address. Same as provided as CGI variable REMOTE_ADDR. As with all IP addresses used for conditional testing this may be wildcard string match or network mask expressed as *address/mask-length* (see Host Addresses).
remote-host:	Client host name if name resolution enabled, otherwise the IP address (same as *remote-addr*). CGI variable REMOTE_HOST.
request:	Detect the presence of specific or unknown request fields. See Request: Keyword.
request-method:	HTTP method (‘‘GET’’, ‘‘POST’’, etc.) specified in the request line. CGI variable REQUEST_METHOD.
request-scheme:	Request protocol as ‘‘http:’’ or ‘‘https:’’. CGI variable REQUEST_SCHEME.
restart:	A numeric value, zero to maximum, representing the number of times path mapping has been SET *map=restart*. Can be used as a boolean or with a digit specified.
script-name:	After the first pass of rule mapping (script component resolution), or during authorization processing, any script component of the request URI.
server-addr:	The service IP address. CGI variable SERVER_ADDR. This may be wildcard string match or network mask expressed as *address/mask-length*.
server_connect_gt:	An integer representing the current server network connections (those currently being processed plus those currently being ‘‘kept alive’’). If greater than this value returns true, otherwise false.
server_process_gt:	An integer representing the current server requests in-progress. If greater than this value returns true, otherwise false.
server-name:	The (possibly virtual) server name. This may or may not exactly match any string provided via the *host* keyword. CGI variable SERVER_NAME.
server-port:	The (possibly virtual) server port number. CGI variable SERVER_PORT.
server-software:	The server identification string, including the version. For example ‘‘HTTPd-WASD/8.0.0 OpenVMS/AXP SSL’’. CGI variable SERVER_SOFTWARE.

Keyword	Description
service:	This is the composite server name plus port as *server-name*:*port*.
ssl:	Simple boolean value. If request is via Secure Sockets Layer then this will be true.
syi_arch_name:	System information; CPU architecture of the server system, ''Alpha'', ''IA64'' or ''VAX''.
syi_hw_name:	System information; hardware identification string, for example ''AlphaStation 400 4/233''.
syi_nodename:	System information; the node name, for example ''KLAATU''.
syi_version:	System information; VMS version string, for example ''V7.3''.
tcpip:	A string derived from the UCX$IPC_SHR shareable image. It looks something like this ''Compaq TCPIP$IPC_SHR V5.1-15 (11-JAN-2001 02:28:33.95)'' and comprises the agent (Compaq, MultiNet, TCPware, unknown), the name of the image, the version and finally the link date.
time:	Compare to current system time. See Time: Keyword.
trnlnm:	Translate a logical name. See Trnlnm: Keyword.
user-agent:	Browser identification string as provided in ''User-Agent:'' request header field. CGI variable HTTP_USER_AGENT.
x-forwarded-for:	Proxied client name or address as provided in ''X-Forwarded-For:'' request header field. CGI variable HTTP_X_FORWARDED_FOR.

Notepad: Keyword

The *request notepad* is a string storage area that can be used to store and retrieve ad hoc information during path mapping and subsequent authorization processing. The notepad contents can be changed using the SET *notepad=<string>* or appended to using SET *notepad=+<string>* (Section 13.4.5). These contents then can be subsequently detected using the *notepad:* conditional keyword (or the obsolescent 'NO' mapping conditional) and used to control subsequent mapping or authorization processing.

Note

Notepad information persists across internal redirection processing (Section 13.4.2) and so may be used when the regenerated request is mapped and authorized. To prevent such information from unexpectedly interfering with internally redirected requests a *notepad=*'' can be used to empty the storage area.

Rand: Keyword

At the commencement of each pass a new pseudo-random number is generated (and therefore remains constant during that pass). The *rand:* conditional is intended to allow some sort of distribution to be built into a set of rules, where each pass (request) generates a different one. The random conditional accepts two parameters, a *modulas* number, which is used to modulas the base number, and a *comparison* number, which is compared to the modulas result.

Hence the following conditional rules

```
if (rand:3:0)
   do this
elif (rand:3:1)
   do this
else
   do this
endif
```

would pseudo-randomly generate base numbers of 0, 1, 2 and perform the appropriate conditional block. Over a sufficient number of usages this should produce a relatively even distribution of numbers. If the modulas is specified as less than two (i.e. no distribution factor at all) it defaults to 2 (i.e. a distribution of 50%). Hence the following example should be the equivalent of a coin toss.

```
if (rand:)
   heads
else
   tails
endif
```

Request: Keyword

Looks through each of the lines of the request header for the specified request field and/or value. This may be used to detect the presence of specific or unknown (to the server) request fields. When detecting a specified just field the name can be provided

```
if (request:"Keep-Alive:*")
```

matching any value, or specific values can also be matched for

```
if (request:"User-Agent:*Opera*")
```

Note that all request fields known to the server have a specific associated conditional keyword (i.e. ‘‘user-agent:’’ for the above example). To determine whether any request fields unknown to the server have been supplied use the *request:* keyword as in the following example.

```
if (request:?)
   map * /cgi-bin/unknown_request_notify.com*
endif
```

Time: Keyword

The *time:* conditional allows server behaviour to change according to the time of day, week, or even year. It compares the supplied parameter to the current system time in one of three ways.

1. The supplied parameter is in the form ''1200-1759'', which should be read as ''twelve noon to five fifty-nine PM'' (i.e. as a time range in minutes, generalized as *hhmm-hhmm*), where the first is the start time and the second the end time. If the current time is within that range (inclusive) the conditional returns true, otherwise false. If the range doesn't look correct false is always returned.

```
if (time:0000-0000)
   it's midnight
elif (time:0001-1159)
   it's AM
elif (time:1200-1200)
   it's noon
else
   it's PM
endif
```

2. If the supplied parameter is a single digit it is compared to the VMS day of the week (1-Monday, 2-Tuesday . . . 7-Sunday).

```
if (time:6 || time:7)
   it's the weekend
else
   it's the working week
endif
```

3. If the supplied string is not in either of the formats described above it is treated as a string match with a VMS comparision time (i.e. *yyyy-mm-dd hh-mm-ss.hh*).

```
if (time:%%%%-05-*)
   it's the month of May
endif
```

Trnlnm: Keyword

The *trnlnm:* conditional dynamically translates a logical name and uses the value. One mandatory and up to two optional parameters may be supplied.

```
trnlnm:logical-name[;name-table][:string-to-match]
```

The *logical-name* must be supplied; without it false is always returned. If just the *logical-name* is supplied the conditional returns true if the name exists or false if it does not. The default *name-table* is LNM$FILE_DEV. When the optional *name-table* is supplied the lookup is confined to that table. If the optional *string-to-match* is supplied it is matched against the value of the logical and the result returned.

Host Addresses

Host names or addresses can be an alpha-numeric string (if DNS lookup is enabled) or dotted-decimal network address, a slash, then a dotted-decimal mask. For example ‘‘131.185.250.0/255.255.255.192’’. This has a 6 bit subnet. It 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. Using the above example the host 131.185.250.250 would be accepted, but 131.185.250.50 would be rejected. Equivalent notation for this rule would be ‘‘131.185.250.0/26’’.

12.3 Examples

The following provides a collection of examples of conditional mapping and authorization rules illustrating the use of wildcard matching, network mask matching and the various formats in which the rules may be blocked.

1. This first example shows an EXEC mapping rule being applied to a path if the request query string contains the string ‘‘example’’.

```
if (query-string:*example*) exec /* /cgi-bin/example/*
```

2. In this example a block of mapping statements is processed if the virtual service of the request matches that in the conditional, otherwise the block is skipped. Note the indentation to help clarify the structure.

```
if (service:the.host.name:80)
   pass /web/* /dka0/the_host_name_web/*
   pass /graphics/* /dka100/graphics/*
   pass * "404 Resource not found."
endif
```

3. This example a series of tests allow a form of case processing where the first to match will be processed and terminate the matching process. In this case if a match does not occur rule processing continues after the *endif*.

```
if (service:the.host.name:80)
   pass /web/* /dka0/the_host_name_web/*
elif (service:next.host.name:80)
   pass /web/* /dka0/next_host_name_web/*
elif (service:another.host.name:80)
   pass /web/* /dka0/another_host_name_web/*
endif
pass /graphics/* /dka100/graphics/*
pass * "404 Resource not found."
```

4. In this (somewhat contrived) example a nested test is used to check (virtual) server name and that the request is being handled via Secure Sockets Layer (SSL) for security. If it is not an informative message is supplied. The *else* and the quotes are not really required but included here for illustration.

```
if (server-name:the.host.name)
   if (scheme:"https")
      pass /secure/* /dka0/the_host_name_web/secure/*
   else
      pass * /dka0/the_host_name_web/secure/only-via-SSL.html
   endif
endif
```

5. This would be another way to accomplish a similar objective to example 4. This uses a *negation* operator to exclude access to successive mappings if not requesting via SSL.

```
if (server-name:the.host.name)
   if (!SSL:)
      pass * /web/secure/only-via-SSL.html
   endif
   pass /secure/* /web/secure/*
   pass /other/* /web/other/*
   pass /web/* /web/web/*
   pass * "404 Resource not found."
endif
```

6. This example shows the use of a compound conditional using the AND and OR operators. It also illustrates the use of a network mask. It will exclude all access to the specified path unless the request is originating from within a specified network (perhaps an intranet) or via SSL.

```
if (path:/sensitive/* && !(remote-addr:131.185.250.0/24 || SSL:))
   pass * 404 "Access denied (SSL only)."
endif
```

7. This example illustrates restricting authentication to SSL.

```
[[*]]
["Your VMS password"=VMS]
if (!request-scheme:https)
   * r+w,#0
endif
```

8. Logical name translation may be used to dynamically alter the flow of rule interpretation.

```
if (trnlnm:HTTPD_EXAMPLE)
   pass /* /example/*
else
   pass /* /*
endif
```

Chapter 13
Mapping Rules

Mapping rules are used for a few different purposes.

1. To map a request *path* onto the VMS file system.
2. To process a request path according to specified criteria resulting in an effective path that is different to that supplied with the request.
3. To identify requests requiring script activation and to parse the script from the path portion of that request. The path portion is then independently re-mapped.
4. To conditionally map to different end-results based on one or more criteria of the request.
5. To provide differing virtual sites depending on the actual service accessed by the client.

Mapping is basically for server-internal purposes only. The only time the path information of the request itself is modified is when a script component is removed. At all other times the path information remains unchanged. Path authorization is always applied to the path supplied with the request.

By default, the system-table logical name HTTPD$MAP locates a common mapping rule file, unless an individual rule file is specified using a job- table logical name. Simple editing of the mapping file changes the rules. Comment lines may be included by prefixing them with the hash ‘‘#’’ character. Although, there is no fixed limit on the number of rules there are the processing implications of scanning a large, linear database.

The [IncludeFile] is a directive common to all WASD configuration, allowing a separate file to be included as a part of the current configuration (Section 6.12).

Rules are given a basic consistency check when loaded (i.e. server startup, map reload, etc.) If there is an obvious problem (unknown rule, missing component, etc., path not absolute) a warning message is generated and the rule is not loaded into the database. This will not cause the server startup to fail. These warning messages may be found in the server process log.

Checking Mapping Rules

A server’s currently loaded mapping rules may be interrogated. See Chapter 18 for further information. Also the Server Administration facility allows realm and arbitrary

paths to be checked against the rule database in real-time using the WATCH facility (Chapter 19). In this way the rule database may be checked against test or even live requests.

Any subsequent changes to the mapping file may be reloaded into the running HTTPd server using the following command on the server system:

```
$ HTTPD /DO=MAP
```

13.1 Rule Interpretation

The rules are scanned from first towards last, until a matching final rule is encountered (PASS, EXEC, SCRIPT, FAIL, REDIRECT, UXEC and USER) when the mapping pass concludes. Non-final rules (MAP and SET) perform the appropriate action and continue to the next rule. One, two or more passes through the rules may occur due to implicit processing (if the path contains a script component) or by explicit restart (SET *map=restart*).

String Matching

The basis of path mapping is string pattern matching, comparing the request specified path, and optionally other components of the request when using configuration conditionals (Chapter 12), to a series of patterns, usually until one of the patterns matches, at which stage some processing is performed. Both wildcard and regular expression based pattern matching is available. All rules have a *template* (string pattern to match against the path). Some rules have a *result* (how to restructure the components matching from the template).

- The **template** may contain one or more asterisk (''*'') wildcard symbols, or a regular expression with optional grouping operators. This is pattern matched against the request path (Chapter 11). If neither is present then the path must match the *template* exactly.
- The **result** may contain one or more asterisk (''*'') substitution symbols. The *result* wildcards are expanded to replace the matching strings of the respective *template* wildcards or pattern groups. Specified wildcard substitution is available (Section 11.4). Characters represented by wildcards in the *template* not represented by a corresponding wildcard in the *result* are ignored. Non-wildcard *result* characters are directly inserted in reconstructed path. Non-wildcard characters in the *template* are ignored. If the *result* contains no wildcards it completely replaces the URL path.

Virtual Servers

As described in Section 6.3 virtual service syntax may be used with mapping rules to selectively apply rules to one specific service. If virtual services are configured rule interpretation sees only rules common to all services and those specific to it's own service (host address and port). In all other aspects rule interpretation applies as described above.

Processing Overhead

Naturally, each rule that needs to be processed adds a little to consumed CPU, introduces some latency, and ultimately reduces throughput. The test-bench has shown this to be acceptably small compared to the overall costs of responding to a request. Using the ApacheBench tool on a Digital AlphaStation 500/333 with 256MB, VMS V7.3-1, and WASD v8.3, with a simple access to /ht_root/exercise/0k.txt showed approximately 108 requests/second throughput using the following mapping file.

```
pass /ht_root/exercise/*
```

After adding various quantities of the same intervening rule

```
pass /ht_root/example/*
pass /ht_root/example/*
   .
   .
   .
pass /ht_root/example/*
pass /ht_root/exercise/*
```

the following results were derived.

Mapping Overhead

Intervening Rules	Requests/S	Throughput
0	108	baseline
100	104	-3.7%
200	100	-7.4%
500	91	-15.7%
1000	70	-35.1%

Although this is a fairly contrived set-up and actual real-world rule-sets are more complex than this, even one hundred rules is a very large set, and it does indicate that for all intents and purposes mapping rules may be used to achieve desired objectives without undue concern about impact on server throughput.

13.2 VMS File System Specifications

The VMS file system in mapping rules is always assumed to begin with a device or concealed device logical. Specifying a Master File Directory (MFD) component, the [000000] is completely optional, although always implied. The mapping functions will always insert one if required for correct file system syntax. That is, if the VMS file system mapping of a path results in a file in a top-level directory an MFD is inserted if not explicitly present in the mapping. For example, both of the following paths

```
/dka100/example.txt
/dka100/000000/example.txt
```

would result in a mapping to

```
DKA100:[000000]EXAMPLE.TXT
```

The MFD is completely optional when both specifying paths in mapping rules and when supplying paths in a request. Similarly, when supplying a path that includes directory components, as in

```
/dka100/dir1/dir2/example.txt
/dka100/000000/dir1/dir2/example.txt
```

both mapping to

```
DKA100:[DIR1.DIR2]EXAMPLE.TXT
```

Implication

When using logical names in file system mappings they must be able to be used as concealed devices and cannot be logical equivalents of directory specifications.

Concealed device logicals are created using the following syntax:

```
$ DEFINE LOGICAL_NAME device:[dir1.dir2.]
$ DEFINE LOGICAL_NAME /TRANSLATION=CONCEALED physical_device:[dir1.dir2.]
```

For ODS-2 volumes (Section 13.3 immediately below), when during rule mapping of a path to a VMS file specification an RMS-invalid character (e.g. ‘‘+’’) or syntax (e.g. multiple periods) is encountered a dollar symbol is substituted in an attempt to make it acceptable. This functionality is often useful for document collections imported to the local web originating from, for instance, a Unix site that utilizes non-RMS file system syntax. The default substitution character may be changed on a per-path basis using the SET rule (Section 13.4.5).

13.3 Extended File Specifications (ODS-5)

OpenVMS Alpha V7.2 introduced a new on-disk file system structure, ODS-5. This brings to VMS in general, and WASD and other Web servers in particular, a number of issues regarding the handling of characters previously not encountered during (ODS-2) file system activities. It is necessary to distinguish paths to ODS-5, extended specification volumes from the default ODS-2 ones (Section 13.4.5).

13.3.1 Characters In Request Paths

There is a standard for characters used in HTTP requests paths and query strings (URLs). This includes conventions for the handling of reserved characters, for example ‘‘?’’, ‘‘+’’, ‘‘&’’, ‘‘=’’ that have specific meanings in a request, characters that are completely forbidden, for example white-space, control characters (0x00 to 0x1f), and others that have usages by convention, for example the ‘‘~’’, commonly used to indicate a username mapping. The request can otherwise contain these characters provided they are URL-encoded (i.e. a percentage symbol followed by two hexadecimal digits representing the hexadecimal-encoded character value).

There is also an RMS standard for handling characters in extended file specifications, some of which are forbidden in the ODS-2 file naming conventions, and others which have a reserved meaning to either the command-line interpreter (e.g. the space) or the file system structure (e.g. the ‘‘:’’, ‘‘[’’, ‘‘]’’ and ‘‘.’’). Generally the allowed but reserved characters can be used in ODS-5 file names if escaped using the ‘‘^’’ character. For example, the ODS-2 file name ‘‘THIS_AND_THAT.TXT’’ could be named ‘‘This^_^&^_That.txt’’ on an ODS-5 volume. More

complex rules control the use of character combinations with significance to RMS, for instance multiple periods. The following file name is allowed on an ODS-5 volume, ‘‘A-GNU-zipped-TAR-archive^.tar.gz’’, where the non-significant period has been escaped making it acceptable to RMS.

The WASD server will accept request paths for file specifications in both formats, URL-encoded and RMS-escaped. Of course characters absolutely forbidden in request paths must still be URL-encoded, the most obvious example is the space. RMS will accept the file name ‘‘This^ and^ that.txt’’ (i.e. containing escaped spaces) but the request path would need to be specified as ‘‘This%20and%20that.txt’’, or possibly ‘‘This^%20and^%20that.txt’’ although the RMS escape character is basically redundant.

Unlike for ODS-2 volumes, ODS-5 volumes do not have ‘‘invalid’’ characters, so no processing is performed to ensure RMS compliance.

13.3.2 Characters In Server-Generated Paths

When the server generates a path to be returned to the browser, either in a viewable page such as a directory listing or error message, or as a part of the HTTP transaction such as a redirection, the path will contain the URL-encoded equivalent of the *canonical form* of an extended file specification escaped character. For example, the file name ‘‘This^_and^_ that.txt’’ will be represented by ‘‘This%20and%20that.txt’’.

When presenting a file name in a viewable page the general rule is to also provide this URL-equivalent of the unescaped file name, with a small number of exceptions. The first is a directory listing where VMS format has been requested by including a version component in the request file specification. The second is in similar fashion, but with the *tree* facility, displaying a directory tree. The third is in the navigation page of the *UPDate* menu. In all of the instances the canonical form of the extended file specification is presented (although any actual reference to the file is URL-encoded as described above).

13.4 Rules

These are the categories of mapping rules.

- Map paths to the file system, and to other paths:
  MAP
  PASS
  FAIL
  REDIRECT
  USER
- Provide access to scripting:
  EXEC
  SCRIPT
  UXEC
- Sets characteristics against particular paths:
  SET

13.4.1 MAP, PASS, FAIL Rules

1. **map** ***template result***

   If the URL path matches the template, substitute the *result* string for the path and use that for further rule processing. Both template and result paths must be absolute (i.e. begin with ''/'').

2. **pass** ***template***
   pass ***template result***
   pass ***template ''999 message text''***

   If the URL path matches the template, substitute the result if present (if not just use the original URL path), processing no further rules.

   The *result* should be a either a physical VMS file system specification in URL format or an *HTTP status-code message* (see below). If there is a direct correspondance between the *template* and *result* the result may be omitted.

   Note

   The PASS directive is also used to *reverse-map* VMS file specifications to the URL path format equivalent.

   An HTTP status-code message can be provided as a result. The server then generates a response corresponding to that status code containing the supplied message. Status-code results should be enclosed in one of single or double quotes, or curly braces. See examples. A 3*nn* status results in a redirection response with the message text comprising the location. Codes 4*nn* and 5*nn* result in an error message. Other code ranges (e.g. 0, 1*nn*, 2*nn*, etc.) simply cause the connection to be immediately dropped, and can be used for that purpose (i.e. no indication of why!)

3. **fail** ***template***

   If the URL path matches the template, prohibit access, processing no further rules. The template path must be absolute (i.e. begin with ''/'').

13.4.2 REDIRECT Rule

1. **redirect** ***template result***

   If the URL path matches the template, substitute the *result* string for the path. Process no further rules. Redirection rules can provide result URLs in one of a number of formats, each with a slightly different behaviour.

   1. The *result* can be a full URL (''http://host.domain/path/to/whatever''). This is used to redirect requests to a specific service, usually on a another host. A *result* may or may not contain a fixed query string (''/path/to/whatever?one=two'').
   2. If the scheme (e.g. ''http:'') is omitted the scheme of the current request is substituted. This allows HTTP requests to be transparently redirected via HTTP and HTTPS (SSL) requests via HTTPS (e.g. ''//host.domain/path/to/whatever'', note the leading double-slash).

3. In a similar fashion both the scheme and the host name may be omitted (e.g. ''///path/to/whatever'', note the leading triple-slash). The server then substitutes the appropriate request scheme and host name before returning the redirection to the client.

4. If the scheme is provided but no host component the current request's host information is substituted and the redirection made using that (e.g. ''https:///secure/path/to/whatever''. This effectively allows a request to be redirected from standard to SSL, or from SSL to standard HTTP on the same server.

5. Alternatively, it may be just a path (''/path/to/whatever'', a single leading slash), which will cause the server to internally generate an entire new request structure to process the new path (i.e. request redirection is not returned to the client).

Note

Internal redirection (as this is termed) is a fundamental mechanism available with WASD to completely change the request path and/or query string components for the request - transparently to the client. It is essentially a complete rewrite of the request.

6. Only if the last character in the *result* is a question mark (''?'') will any query string in the original be propagated into the redirection URL (that is the original request ''/original/test.txt?plus=query'' is mapped using ''redirect /original/* /path/to/*?'' does the resulting URL become ''/path/to/test.txt?plus=query'').

13.4.3 USER Rule

The USER rule maps a VMS user account default device and directory (i.e. *home* directory) into a request path. That is, the base location for the request is obtained from the VMS systems SYSUAF file. This is usually invoked by a request path in the form ''/~username/'', see Section 13.8 for more detailed information.

1. **user** ***template result***

   If the path matches the template then the result is substituted, with the following conditions. At least one wildcard must be present. The first wildcard in the result substitutes the username's home directory into the path (in place of the ''~username''). Any subsequent wildcard(s) substitute corresponding part(s) of the original path.

   If the user DANIEL's default device and directory were

   ```
   USER$DISK:[DANIEL]
   ```

   the following rule

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

   would result in the following path being mapped and used

   ```
   /user$disk/daniel/www/
   ```

Note

Accounts that possess SYSPRV, are CAPTIVE, have been DISUSERED or that have expired passwords will not be mapped. A ''directory not found'' error report is returned.

13.4.4 EXEC/UXEC and SCRIPT, Script Mapping Rules

Also see ''Scripting Environment'' document for further information.

The EXEC/UXEC and SCRIPT directives have the **variants EXEC+/UXEC+ and SCRIPT+**. These behave in exactly the same fashion and simply mark the rule as representing a CGIplus script environment.

The EXEC/UXEC rules maps script **directories**.

The SCRIPT rules maps script **file names**. It behaves a little differently to the EXEC rule, essentially supplying in a single rule the effect of a MAP then an EXEC rule.

Both rules must have a *template* and *result*, and both must end in a wildcard asterisk. The placement of the wildcards and the subsequent functionality is slightly different however. Both template and result paths must be absolute (i.e. begin with ''/'').

1. **exec *template result***

   The EXEC rule requires the *template*'s asterisk to immediately follow the slash terminating the directory specification containing the scripts. The script name follows immediately as part of the wildcard-matched string. For example:

   ```
   exec /htbin/* /ht_root/script/*
   ```

   If the URL path matches the template, the result, including the first slash-terminated part of the wildcard-matched section, becomes the URL format physical VMS file specification the script to be executed. What remains of the original URL path is used to create the path information. Process no further rules.

   Hence, the EXEC rule will match multiple script specifications without further rules, the script name being supplied with the URL path. Hence any script (i.e. procedure, executable) in the specified directory is accessable, a possible security concern if script management is distributed.

2. **exec *template (run-time-environment)result***

   A variation on the ''exec'' rules allows a Run-Time Environment (RTE) to be mapped. An RTE is a persistant scripting environment not unlike CGIplus. The essential difference is an RTE provides an environment in which a variety of scripts can be run. It is often an interpreter, such as Perl, where the advantages of persistance (reduced response latency and system impact) are available. For more information on RTEs and how they operate see the ''WASD Scripting Environment'' document.

   The RTE executable is specified in parentheses prefixed to the mapping result, as show in this example:

   ```
   exec /pl-bin/* (cgi-bin:[0000000]perlrte.exe)/ht_root/src/perl/*
   ```

3. **script *template result***

   The SCRIPT rule requires the *template*'s asterisk to immediately follow the *unique string* identifying the script in the URL path. The wildcard-matched string is the following path, and supplied to the script. For example:

   ```
   script /conan* /ht_root/script/conan*
   ```

If the URL path matches the template, the result becomes the URL format physical VMS file specification for the DCL procedure of the script to be executed (the default file extension of ‘‘.COM’’ is not required). What remains of the original URL path is used to create the path information. Process no further rules.

Note

The wildcard asterisk is best located immediately after the unique script identifier. In this way there does not need to be any path supplied with the script. If even a slash follows the script identifier it may be mapped into a file specification that may or may not be meaningful to the script.

Hence, the SCRIPT rule will match only the script specified in the *result*, making for finely-granular scripting at the expense of a rule for each script thus specified. It also implies that only the script name need precede any other path information.

It may be thought of as a more efficient implementation of the equivalent functionlity using two CERN rules, as illustrated in the following example:

```
map /conan* /script/conan*
exec /cgi-bin/* /cgi-bin/*
```

4. **uxec** ***template result***

   The UXEC rule is an analog to the EXEC rule, except it is used to map user scripts. It requires two mapping asterisks, the first for the username, the second for the script name. It must be used in conjunction with a SET *script*=*as*=~ rule. For example:

   ```
   SET   /~*/www/cgi-bin/*  script=as=~
   UXEC  /~*/cgi-bin/*  /*/www/cgi-bin/*
   ```

   For further information see User Account Scripting and the ‘‘Scripting Overview, Introduction’’.

Script Location

It is conventional to locate script images in HT_ROOT:[AXP-BIN] or HT_ROOT:[VAX-BIN] (depending on the platform), and procedures, etc. in HT_ROOT:[CGI-BIN]. These multiple directories are accessable via the single search list logical CGI-BIN.

Script files can be located in area completely outside of the HT_ROOT tree. Two approaches are available.

1. Modify the search list CGI-BIN to include the additional directories. Only should be done with extreme care.
2. Use mapping rules to make the script accessable. This can be done by using the EXEC or SCRIPT rule to specify the directory directly as in these examples

   ```
   exec /mycgi-bin/* /site_local_scripts/bin/*
   script /myscript* /web/myscripts/bin/myscript.exe*
   ```

   or by using the MAP rules to make a hierarchy of script locations obvious and accessable, as in this example

   ```
   map /cgi-bin/myscripts/* /cgi-bin_myscripts/*
   exec /cgi-bin_myscripts/* /web/myscripts/bin/*
   ```

EXEC Directories and EXEC Files

Generally directories are specified as locations for script files. This is the more common application, with the EXEC rules used as in this example

```
exec /cgi-bin/* /cgi-bin/*
```

Mapping a file type into an EXEC behaviour is also supported. This allows all files within the specified path and with the matching file suffix (extension) to be activated as scripts. Of course a script runtime must be available for the server to be able activate it. The following example demonstrates mapping all files ending in .CGI in the /web/ tree as executable scripts.

```
exec /web/*.cgi* /web/*.cgi*
```

WARNING

Remember scripts are **executables**. Enabling scripting in a general user area allows **any** user to write and execute any script, by default under the server account. Deploy with discretion.

13.4.5 SET Rule

The SET rule does not change the mapping of a path, it just sets one or more characteristics against that path that affect the subsequent processing in some way. It is a general purpose rule that conveniently allows the administrator to tell the server to process requests with particular paths in some ad hoc and generally useful fashion. Most SET parameters are single keywords that act as boolean switches on the request, some require parameter strings. Multiple space-separated parameters may be set against against the one path in a single SET statement.

- **ACCEPT=LANG=<*parameter*> -** Allows a path to be marked for language-variant document processing.
  - ''ACCEPT=LANG=DEFAULT=*language*'' sets the default language
  - ''ACCEPT=LANG=CHAR=*character*'' sets the delimiting character
  - ''ACCEPT=LANG=VARIANT=*name*|*type*'' allows the alternate file-type variant to be specified
  - ''ACCEPT=LANG=(DEFAULT=*language*,CHAR=*character*)'' sets both (etc.)
  - ''NOACCEPT=LANG'' disables language variant processing (on a subtree for example)

  For detailed configuration information see Section 6.7.
- **ALERT[=<*keyword*>] -** Marks a path as being of specific interest. When a request containing this path is detected by the server it puts a message into the the server process log and perhaps of greater immediate usefulness the increase in alert hits is detected by HTTPDMON and this (optionally) provides an audible alert. The following is ordered according to how early in processing the alert is signalled.
  - ''ALERT=MAP'' generates this alert immediately after path mapping (i.e. before the request actually begins being processed).
  - ''ALERT=AUTH'' after authorization (i.e. when any remote username has been resolved).

- ''ALERT=END'' at the conclusion of process (the default).
- ''ALERT=*integer*'' if the response HTTP status matches the specific integer.
- ''NOALERT'' cancels alerts on this path (perhaps subpath).

- **AUTH=<*keyword*> -** Changes the specified characteristic during subsequent authorization processing.
  - ''[NO]AUTH=ALL'' All requests matching this path must have been subject to authorization or fail with a forbidden status. This is a per-path requivalent of implementing the per-server /AUTHORIZE=ALL policy (Section 15.13), and is a little ''belt and braces'' in a certain sense, but does permit a site to further avoid unintended information leakage (in this case through the failure ensure a given path has authorization).
  - ''[NO]AUTH=ONCE'' If a request path contains both a script component and a resource component by default the WASD server makes sure both parts are authorized before allowing access (Chapter 15). This can be disabled using this path setting. When this is done only the original request path undergoes authorization.
  - ''AUTH=REVALIDATE=<*hh:mm:ss*>'' Authorization is cancelled and the client requested to reenter the username and password if this period expires between authorized requests. Overrides configuration directive [AuthRevalidateUserMinutes].
  - ''AUTH=SYSUAF=PWDEXPURL=<*string*>'' Parallels the [AuthSysUafPwdExpURL] configuration directive, allowing it to be set on a per-path or virtual service basis (Section 15.14).
- **CACHE=<*keyword*> -** The default is to cache files (when caching is enabled, Chapter 20).
  - ''CACHE=NONE'' disables caching of files matching this rule
  - ''CACHE=EXPIRES=0'' cancels previous mapped expiry
  - ''CACHE=EXPIRES=DAY'' expires on change of day
  - ''CACHE=EXPIRES=HOUR'' expires on change of hour
  - ''CACHE=EXPIRES=MINUTE'' expires on change of minute
  - ''CACHE=EXPIRES=<*period*>'' sets the expiry period for the entry
  - ''CACHE=GUARD=<*period*>'' sets the guard period (no reload) for the cache entry
  - ''CACHE=MAX=<*integer*>'' cache files up to this many kilobytes (overrides [CacheFileKBytesMax]))
  - ''CACHE=[NO]CGI'' cache CGI-compliant (script) responses
  - ''CACHE=[NO]FILE'' cache files matching this rule (the default)
  - ''CACHE=[NO]NET'' cache any network output
  - ''CACHE=[NO]NPH'' cache NPH (non-parse-header script) responses
  - ''CACHE=[NO]SCRIPT'' cache both CGI and NPH responses
  - ''CACHE=[NO]SSI'' cache SSI document responses
  - ''CACHE=[NO]QUERY'' cache (script) regardless of containing a query string

- ‘‘CACHE=[NO]PERM’’ permanently cache these files

- **CGIPLUSIN=*<keyword>* -** Provides control over how CGIplus records on the CGIPLUSIN stream are carriage controlled and how the stream is terminated. A little esoteric certainly; ask Alex Ivanov ;^)
  - ‘‘CGIPLUSIN=CC=NONE’’ no carriage control
  - ‘‘CGIPLUSIN=CC=LF’’ each record has a trailing line feed (0x0a)
  - ‘‘CGIPLUSIN=CC=CR’’ a trailing carriage return (0x0d)
  - ‘‘CGIPLUSIN=CC=CRLF’’ a trailing line feed then carriage return (0x0d0a)
  - ‘‘CGIPLUSIN=[NO]EOF’’ the end of the record stream is indicated using an end-of-file
- **CGIPREFIX=*<string>* -** CGI environment variable names are by default prefixed with ‘‘WWW_’’. This may be changed on a per-path basis using this SET rule. To remove the prefix altogether for selected scripts use ‘‘CGIprefix=’’.
- **CHARSET=*<string>* -** This setting allows overriding of the server default ([CharsetDefault] configuration parameter) content-type character set (in the response header) for text files (plain and HTML). A string is required as in the following example, ‘‘charset=ISO-8859-5’’.
- **CONTENT=*<string>* -** The content-type of a file is normally determined by the file’s type (extension). This setting allows files matching the template to be returned with the specified content-type. The content-type must be specified as a parameter, e.g. ‘‘content=application/binary’’.
- **DIR=*<keyword>* -** Allows directory listing to be controlled on a per path basis. These parallel the coresponding configuration [Dir..] directives.
  - ‘‘DIR=[NO]ACCESS’’ allows directory listing
  - ‘‘DIR=ACCESS=SELECTIVE’’ allows directory listing if the directory contain the file .WWW_BROWSABLE
  - ‘‘DIR=[NO]IMPLIEDWILDCARD’’ add wildcards if not in path
  - ‘‘DIR=STYLE=’’ set the style of a directory listing
    - ‘‘ANCHOR’’ the current and default WASD style (post-v8.2)
    - ‘‘DEFAULT’’ the current WASD style (post-v8.2)
    - ‘‘ORIGINAL’’ the traditional (pre-v8.2)
    - ‘‘HTDIR’’ Alex Ivanov’s HTdir style
  - ‘‘DIR=[NO]WILDCARD’’ allow a directory listing to be ‘‘forced’’ by including wildcards in the path

  For detailed configuration information see Section 6.7.
- **[NO]EXPIRED -** This setting allows files in the specified paths to be sent pre-expired. The browser should always then reload them whenever accessed.

- **HTML=<*keyword*>=<*string*> -** Allows the <BODY> tag, and header and/or footer characteristics and text to be added to selected server generated pages such as directory listings and error messages.
  - ''HTML=BODYTAG='' specifies the page <BODY> tag characteristics (e.g. html=bodytag=''BGCOLOR=#ffffff'')
  - ''HTML=HEADER='' the page header text
  - ''HTML=HEADERTAG='' the <TD> tag characteristics of the header table (e.g. html=headertag=''BGCOLOR=#cccccc'')
  - ''HTML=FOOTER='' the page footer text
  - ''HTML=FOOTERTAG='' the <TD> tag characteristics of the footer table

  The *headertag* and *footertag* directives also allow the full table tag to be specified, allowing greater flexibility with these parts of the page (e.g. html=footertag=''<TABLE BORDER=1 CELLPADDING=10 CELLSPACING=0><TR><TD BGCOLOR=#cccccc>''.
- **HTTP=<*parameter*> -** Explicitly sets an aspect of the HTTP request header.
  - ''HTTP=ACCEPT-CHARSET=<*string*>'' the ''Accept-Charset:'' field
  - ''HTTP=ACCEPT-LANGUAGE=<*string*>'' the ''Accept-Language:'' field
- **INDEX=<*string*> -** This setting provides the ''Index of'' (directory listing) format string for directory paths matching the template. It uses the same formatting as can be supplied with a URL and overrides any query string passed via any URL.
- **[NO]LOG -** When server access logging is enabled the default is to log all requests. The NOLOG setting suppresses logging for requests involving the specified path template.
- **MAP=<*parameter*> -** Controls aspects of the mapping processing itself (from that point in the rules onwards of course).
  - ''[NO]MAP=ELLIPSIS'' By default the use of the VMS file specification ellipsis wilcard (''...'') is not allowed. This enables this for the path specified. Use with caution.
  - ''[NO]MAP=ONCE'' Normally, when a script has been identified during mapping, the resultant path information is also mapped in a second pass. This can be suppressed by SETting the path as MAP=ONCE. The resultant path is then given to the script without further processing.
  - ''MAP=RESTART'' Causes an immediate change to the order of rule processing. Instead of the next rule, the first rule in the configuration is processed. This is intended to remove the need for copious repetition in the rule set. A common or set of common processing blocks can be established near the start of the rule set and be given requests from processing from points further down in the rules. It is intended to be used only once or perhaps twice and will abort the request if it occurs too often. Can be detected using the *restart:* conditional (Section 12.2). Use with caution! Injudicious use would make unexpected mappings expected!
  - ''[NO]MAP=ROOT=<*string*>'' Prefixes the results of following rules with the specified path so that they are all subordinate to it. This also populates the DOCUMENT_ ROOT CGI variable. See Document Root.

- ''[NO]MAP=SET=IGNORE'' All path SETings following an IGNORE are completely ignored (not applied to the mapping or request characteristics) until a subsequent NOINGORE is encountered.
- ''[NO]MAP=SET=REQUEST'' All path SETings following a NOMAP=SET=REQUEST are only applied to the mapping and not to the request's characteristics until a subsequent MAP=SET=REQUEST is encountered. Intended for use during callouts. These can be detected using the *callout:* conditional (Section 12.2).

- **NOTEPAD=[+]<*string*> -** The *request notepad* is a string storage area that can be used to store and retrieve ad hoc information during path mapping and subsequent authorization processing. Multiple *notepad=string* set against the one request override previous settings unless preceded by a leading plus symbol, when it appends. These contents then can be subsequently detected using the *notepad:* conditional keyword (Notepad: Keyword) or the obsolescent 'NO' mapping conditional.
- **ODS=<*keyword*> -** Directs the server on how to process file names for naming conventions other than ODS-2 (the default). Be sure to add an asterisk at the end of the specific ODS path otherwise only the top-level will set!
  - ''ODS=2'' is basically redundant, because if a path is not indicated as anything else it is assumed to be ODS-2. This can be used for clarity in the mapping rules if required.
  - ''ODS=5'' is used to indicate that a particular path maps to files on an ODS-5 (EFS) volume and so the names may comply to extended specifications. This changes the way file names are processed, including for example the replacement of invalid RMS characters (see below).
  - ''ODS=ADS'' is used to process file names that are encoded using the Advanced Server (PATHWORKS 6) schema.
  - ''ODS=PWK'' is used for processing file names encoded using the PATHWORKS 4/5 schema.
  - ''ODS=SMB'' is a synonym for ODS=ADS and makes clear the path is also being served by Samba.
  - ''ODS=SRI'' for file names encoded using the SRI schema (used by MultiNet and TCPware NFS, FTP and other utilities).
- **QUERY-STRING=<*string*> -** Set the request's query string to that specified in the directive. Overloads any current query string. Specify URL-encoded if the characters require it.
- **PROXY=<*parameter*> -** Sets an aspect of proxy request processing.
  - ''PROXY=BIND=<*IP-address*>'' makes outgoing proxy requests appear to originate from this IP address. Must be an address that the media can be bound to.
  - ''PROXY=CHAIN=<*host:port*>'' makes outgoing proxy requests chain to this up-stream proxy server.

- ‘‘PROXY=FORWARDED’’ controls generatation a proxy ‘‘Forwarded:’’ request field. This optional field contains information on the proxy server and as a further option the client name or IP address.
  - ‘‘PROXY=NOFORWARDED’’ disables
  - ‘‘PROXY=FORWARDED[=BY]’’ contains the *by* component.
  - ‘‘PROXY=FORWARDED=FOR’’ contains *by* and the *for* components (client host name)
  - ‘‘PROXY=FORWARDED=ADDRESS’’ contains *by* and the *for* components (client host address)
- ‘‘PROXY=REVERSE=LOCATION=string’’ rewrites the matching ‘‘Location:’’ header field URL of a 302 response from an internal, reverse-proxied server.
- ‘‘PROXY=REVERSE=[NO]VERIFY’’ sets a specialized authorization capability. See HT_ROOT:[SRC.HTTPD]PROXYVERIFY.C for further information.
- ‘‘PROXY=XFORWARDEDFOR=’’ controls generation of a proxy ‘‘X-Forwarded-For:’’ request field. This optional field (a defacto standard originally from the *Squid* caching package) contains the name or IP address of the proxied client.
  - ‘‘PROXY=NOXFORWARDEDFOR’’ disables
  - ‘‘PROXY=XFORWARDEDFOR[=ENABLED]’’ enables
  - ‘‘PROXY=XFORWARDEDFOR=ADDRESS’’ field contains client host address
  - ‘‘PROXY=XFORWARDEDFOR=UNKNOWN’’ field contains *unknown* for the client host name

- **[NO]PROFILE -** When using the server /PROFILE qualifier enable or disable the authentication profile when assessing access for a specific path.
- **REPORT=<*parameter*> -** This setting allows error and other server-generated reports for any specified path to changed between *detailed* and *basic* (Section 6.9.1).
  - ‘‘REPORT=BASIC’’ include less detail in error message
  - ‘‘REPORT=DETAILED’’ includes more detail
  - ‘‘REPORT=4*nn*=*nnn*’’ maps one 400 class HTTP status to another (to conceal the true origins of some error messages)
- **RMSCHAR=<*character*> -** This setting applies to ODS-2 paths (the default) only. Paths SET as ODS-5 do not have this applied. During rule mapping of a path to a VMS file specification, if an RMS-invalid character (e.g. ‘‘+’’) or syntax (e.g. multiple periods) is encountered a dollar symbol is substituted in an attempt to make it acceptable. This setting provides an alternate substitution character. Any general RMS-valid character may be specified (e.g. alpha-numeric, ’$’, ’-’ or ’_’, although the latter three are probably the only REAL choices). A single character is required as in the following example, ‘‘RMSchar=_’’.

- **RESPONSE=HEADER=<*parameter*> -** changes the way in which a response header is generated by the server.
  - ''RESPONSE=HEADER=BEGIN'' suppresses the response header terminating empty line so that the file or other resource can supply additional header fields. It, of course, must supply the header-terminating empty line before beginning to supply the response body.
  - ''RESPONSE=HEADER=FULL'' reverts to normal response header generation behaviour.
  - ''RESPONSE=HEADER=NONE'' suppresses the normal response header generation. It is considered the file or other resource contains and will supply the full HTTP response (in a non-parse-header script fashion).
  - ''RESPONSE=HEADER=ADD=<*string*>'' appends the specified string to the response header. Of course the string should be a legitimate HTTP response field and value line. This mapping can be used to add a particular response directive to matching requests.
- **SCRIPT=<*parameter*> -** Provides controls over various aspects of the scripting environment.
  - ''SCRIPT=AS=<*parameter*>'' for non-server account scripting this rule allows the user account to be either explicitly specified or substituted through the use of the tilde character ''~'' or the dollar ''$''. For further detail see the ''Scripting Overview, Introduction''.
  - ''SCRIPT=BIT-BUCKET=<*hh:mm:ss*>'' specifies the period for which a script continues to execute if the client disconnects. Overrides the HTTPD$CONFIG [DclBitBucketTimeout] configuration directive. For further detail see the ''Scripting Overview, Introduction''.
  - ''SCRIPT=COMMAND=<*string*>'' allows additional parameters and qualifiers to be passed to the script activation command line. First parameter must be an asterisk to use the server resolved script command. If the first parameter is not an asterisk it substitues for the script activation verb. Subsequent parameters must be as they would be used on the command line. The following setting

    ```
    set /cgi-bin/example* script=command="* /ONE /TWO=THREE FOUR"
    ```

    would result in the hypothetical script being command-line activated

    ```
    $ EXAMPLE /ONE /TWO=THREE FOUR
    ```
  - ''SCRIPT=CPU=<*hh:mm:ss*>'' specifies that the server should not allow the script to use more than the specified quantity of CPU time. This is approximate, due to the way the server administers scripting. It can serve to prevent scripts from consuming indefinite quantities of system resources.
  - ''SCRIPT=DEFAULT=<*string*>'' sets the default directory for the script environment (a SET DEFAULT immediately prior to script activation). This can be suppressed (for backward compatibility purposes) using a ''#'' as the target directory. This string is reflected in CGI variable SCRIPT_DEFAULT so that CGIplus script and RTE engines can be informed of this setting for a particular script's environment. Unix

syntax paths may also be specified. If the default begins with a ''/'' character the SET DEFAULT is not performed but the SCRIPT_DEFAULT variable is set appropriately allowing the equivalent of a *chdir( )* to be performed by the scripting environment.

- ''[NO]SCRIPT=FIND'' by default the server always confirms the existance and accessability of a script file by searching for it before attempting to activate it. If it does not exsist it reports an error. It may be possible a Run-Time Environment (RTE) may require to access it's own script file via a mechanism available only to itself. The server script search may be disabled by SETting the path as *nofind*, for example ''script=nofind''. The script path and filename is directly passed to the RTE for it to process and activate.
- ''SCRIPT=PARAM=*(<name=value>)*'' allows non-CGI environment variables to be associated with a particular script path. The name component becomes a variable containing the specified value passed to the script. Multiple, comma-separated *name=value* pairs may be specified. The value may be quoted. The following path setting

```
set /cgi-bin/example* script=params=(first=one,second="Two (and Three)")
```

would result in additional CGI variables available to the script

```
WWW_FIRST == "one"
WWW_SECOND == "Two (and Three)"
```

Multiple *script=params* set against the one request override previous settings unless the parameters are specified with a leading plus symbol, as in

```
set /cgi-bin/example* script=params=+(third=three,fourth="number 4")
```

For futher information see the ''Scripting Overview, CGI''.

- ''[NO]SCRIPT=PATH=FIND'' directs the server to check for and report if the file specified in the path does not exist before activating the script process. Normally this would be left up to the script.
- ''[NO]SCRIPT=QUERY=NONE'' saves a small amount of overhead by suppressing the decomposition of any query string into key or form fields for those environments that do this for themselves.
- ''[NO]SCRIPT=QUERY=RELAXED'' normally when the CGI variables are being prepared for a script and the query string is parsed an error is reported if it uses *x-www-form-urlencoded* format and the encoding contains an error. However some scripts use non-strict encodings and this rule allows those scripts to receive the query strings without the server complaining first.

- **[NO]SEARCH=NONE -** Do not activate the automatic document search script for any query strings associated with this path.
- **SSI=*<parameter>* -** Controls aspects of Server-Side Include engine behaviour.
  - ''[NO]SSI=PRIV'' SSI documents cannot contain privileged directives (e.g. <–#exec ... –>) unless owned by SYSTEM ([1,4]) or are in path set as allowing these directives. Use SSI=priv to enable this, NOSSI=priv to disable. **Caution:** these SSI directives are quite powerful, use great care when allowing any particular document author or authors to use them.

- ''SSI=EXEC=<*string*>'' where <string> is a comma-separated list of the #dcl parameters permitted for the path allows fine-grained control of what capabilities are enabled. The parameter ''#'' enables SSI on a per-path basis.

```
ssi=exec=say,show
ssi=exec=#
```

- **SSLCGI=<*keyword*> -** Enables and sets the type of CGI variables used to represent a Secure Sockets Layer (SSL) CGI variables.
  - ''NOSSLCGI'' disables the facility
  - ''SSLCGI=none'' disables the facility
  - ''SSLCGI=Apache_mod_SSL'' provides Apache mod_ssl style variables
  - ''SSLCGI=Purveyor'' provides Purveyor style variables

  When enabling these variables it is advised to increase the HTTPD$CONFIG [BufferSizeDclCommand] and [BufferSizeCgiPlusIn] directives by approximately 2048.
- **[NO]STMLF -** Specify files to be automatically converted to Stream-LF format. The default is to ignore conversion. STMLF allows selected paths to be converted. See File Record Format.
- **THROTTLE=<*parameter*> -** Controls the concurrent number of scripts being processed on the path.
  - ''THROTTLE=*n,[n,n,n,hh:mm:ss,hh:mm:ss]*''
  - ''THROTTLE=FROM=*n*''
  - ''THROTTLE=TO=*n*''
  - ''THROTTLE=RESUME=*n*''
  - ''THROTTLE=BUSY=*n*''
  - ''THROTTLE=TIMEOUT=QUEUE=*hh:mm:ss*''
  - ''THROTTLE=TIMEOUT=BUSY=*hh:mm:ss*''

  See Section 6.4.
- **TIMEOUT=<*parameter*> -** Sets the appropriate timeout period on a per-path basis.
  - ''TIMEOUT=*hh:mm:ss,hh:mm:ss,hh:mm:ss*''
  - ''TIMEOUT=KEEPALIVE=*hh:mm:ss*''
  - ''TIMEOUT=NOPROGRESS=*hh:mm:ss*''
  - ''TIMEOUT=OUTPUT=*hh:mm:ss*''

  The composite directive has the order *keep-alive* then *no-progress* then *output*. These parallel the respective configuration timeout periods. See Section 8.2.

Of course, as with all mapping rules, paths containing file types (extensions) may be specified so it is quite easy to apply settings to particular groups of files. Multiple settings may be made against the one path, merely separate set directives from each other with white-space. If a setting string is required to contain white-space enclose the string with single or double quotes, or curly brackets. The following example gives a small selection of potential uses.

```
# examples of SET rule usage
# --------------------------
# disable caching for selected paths
set /ht_root/src/* NOcache
set /sys$common/* NOcache
# enable stream-LF conversion in selected directory trees
set /web/* stmlf
set /ht_root/* stmlf
# respond with Cyrillic character set(s) from relevant directories
set /*/8859-5/* charset=ISO-8859-5
set /*/koi8-r/* charset=KOI8-R
# the Sun Java tutorial when UNZIPped contains underscores for invalid characters
set /vms/java/tutorial/* RMSchar=_
# if a request has "/plain-text/" in it's path then ALWAYS return as plain-text!
set /*/plain-text/* content=text/plain
map /*/plain-text/* /*/*
# same for "/binary/"
set /*/binary/* content=text/plain
map /*/binary/* /*/*
# indicate extended file specifications on this path
set /Documents/* ODS=5
pass /Documents/* /ods5_device/Documents/*
# throttle this script's execution, 5 executing, unlimited waiting
set /cgi-bin/big_script* throttle=5
# disable server script search for this RTE
set /onerte/*  script=nofind
exec /onerte/* (CGI-BIN:[000000]ONERTE.EXE)/ht_root/src/one/*
```

Postfix SET Rule

Path SETings may appended to any rule that contains both a template and result. This makes it possible to apply path SETings using matching final rules. For example a matching PASS rule does not require a separate, preceding SET rule containing the same path to also apply required SETings. This is more efficient (requiring less pattern matching) and tends to make the rule set less cluttered.

```
# examples of postfix SET rule usage
# ----------------------------------
# if a request has "/plain-text/" in it's path then ALWAYS return as plain-text!
map /*/plain-text/* /*/* content=text/plain
# same for "/binary/"
map /*/binary/* /*/* content=text/plain
# indicate extended file specifications on this path
pass /Documents/* /ods5_device/Documents/* ODS=5
# throttle this script's execution, 5 executing, unlimited waiting
script /cgi-bin/big_script* throttle=5
```

13.5 Mapping Examples

The example mapping rule file for the WASD HTTP server can be viewed.

online hypertext link

Example of Map Rule

The *result* string of these rules may or may not correspond to to a VMS physical file system path. Either way the resulting rule is further processed before passing or failing.

1. The following example shows a path ‘‘/web/unix/shells/c’’ being mapped to ‘‘/web/software/unix/scripts/c’’, with this being used to process further rules.

```
map /web/unix/* /web/software/unix/*
```

Examples of Pass Rule

1. This example shows a path ‘‘/web/rts/home.html’’ being mapped to ‘‘/user$rts/web/home.html’’, and this returned as the mapped path.

```
pass /web/rts/* /user$rts/web/*
```

2. This maps a path ‘‘/icon/bhts/dir.gif’’ to ‘‘/web/icon/bhts/dir.gif’’, and this returned as the mapped path.

```
pass /icon/bhts/* /web/icon/bhts/*
```

3. This example illustrates HTTP status code mapping. Each of these does basically the same thing, just using one of the three possible delimiters according to the characters required in the message. The server generates a 403 response with has as it’s text the following message. (Also see the conditional mapping examples.)

```
pass /private/* "403 Can't go in there!"
pass /private/* '403 "/private/" is off-limits!'
pass /private/* {403 Can't go into "/private/"}
```

Examples of Fail Rule

1. If a URL path ‘‘/web/private/home.html’’ is being mapped the path would immediately be failed.

```
fail /web/private/*
```

2. To ensure all access fails, other than that explicitly passed, this entry should be included the the rules.

```
fail /*
```

Examples of Exec and Script Rules

1. If a URL path ‘‘/htbin/ismap/web/example.conf’’ is being mapped the ‘‘/ht_root/script/’’ must be the URL format equivalent of the physical VMS specification for the directory locating the script DCL procedure. The ‘‘/web/example.conf’’ that followed the ‘‘/htbin/ismap’’ in the original URL becomes the translated path for the script.

```
exec /cgi-bin/* /cgi-bin/*
```

2. If a URL path ‘‘/pl-bin/example/this/directory/and-file.txt’’ is being mapped the script name and filename become ‘‘/pl-bin/example’’ and ‘‘HT_ROOT:[SRC.PERL]EXAMPLE.PL’’ respectively, the path information and translated become ‘‘/this/directory/and-file.txt’’ and ‘‘THIS:[DIRECTORY]AND-FILE.TXT’’, and the interpreter (run-time environment) activated to interpret the script is CGI-BIN:[000000]PERLRTE.EXE.

```
exec /pl-bin/* (cgi-bin:[000000]perlrte.exe)/ht_root/src/perl/*
```

3. If a URL path ‘‘/conan/web/example.hlb’’ is being mapped the ‘‘/ht_root/script/conan’’ must be the URL format equivalent of the physical VMS specification for the DCL procedure. The ‘‘/web/example.hlb’’ that followed the ‘‘/conan/’’ in the original URL becomes the translated path for the script.

```
script /conan* /ht_root/script/conan*
```

Example of Redirect Rule

1. If a URL path ‘‘/AnotherGroup/this/that/other.html’’ is being mapped the URL would be redirected to ‘‘http://host/this/that/other.html’’

```
redirect /AnotherGroup/* http://host/group/*
```

13.6 Virtual Servers

As described in Section 6.3, virtual service syntax may be used with mapping rules 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 mappings (e.g. for scripting) to be shared.

```
# a mapping rule example of virtual servers
[[alpha.domain.name:80]]
# ALPHA is the only service allowing access to VMS help directory
pass /sys$common/syshlp/*
[[beta.domain.name:80]]
# good stuff is only available from BETA
pass /good-stuff/*
# BETA has it's own error report format, the others share one
pass /errorreport /httpd/-/errorreportalpha.shtml
[[gamma.domain.name:80]]
# gamma responds with documents using the Cyrillic character set
set /* charset=ISO-8859-5
[[*]]
# common file and script mappings
exec /cgi-bin/* /cgi-bin/*
exec+ /cgiplus-bin/* /cgi-bin/*
script+ /help/* /cgiplus-bin/conan/*
pass /errorreport /httpd/-/errorreport.shtml
# now the base directories for all documents
[[alpha.domain.name:80]]
/* /web/alpha/*
[[beta.domain.name:80]]
/* /web/beta/*
[[gamma.domain.name:80]]
/* /web/gamma/*
[[*]]
# catch-all rule (just in case :^)
pass /* /web/*
```

The Server Administration page WATCH report (Section 18.4) provides the capability to view the rule databse as well as rule mapping during actual request processing, using the WATCH facility.

13.7 Conditional Mapping

(Somewhat) Deprecated and Discouraged

There is now a more versatile approach to achieving the same functionality described in this section, see Chapter 12. Conditional mapping will be retained for the foreseeable future with this documentation available for reference by older site configurations. The two approaches may be used concurrently as required.

The purpose of *conditional mapping* is to **apply** rules only after certain criteria other than the initial path match are met.

THIS OFFERS A POWERFUL TOOL TO THE SERVER ADMINISTRATOR!

Conditional mapping can be applied on the following criteria:

client internet address
browser-accepted languages
browser-accepted character sets
browser-accepted content-types
browser identification string
cookie data
host and port specified in request header

HTTP method (GET, POST, etc.)
proxy/gateway host(s) request forwarded by
refering page
request scheme (protocol . . . ‘‘http:’’ or ‘‘https:’’)
query string
server name
server port

Conditionals must follow the rule and are delimited by ‘‘[’’ and ‘‘]’’. Multiple, space-separated conditions may be included within one ‘‘[...]’’. This behaves as a logical OR (i.e. the condition only needs one matched to be true). Multiple ‘‘[...]’’ conditionals may be included against a rule. These act as a logical AND (i.e. all must have at least one condition matched). If a condition begins with a ‘‘!’’ it acts as a negation operator (i.e. matched strings result in a false condition, unmatched strings in a true condition). The result of an entire conditional may also be negated by prefixing the ‘‘[’’ with a ‘‘!’’.

If a conditional, or set of conditionals, is not met the rule is completely ignored.

Both wildcard and regular expression pattern matching is available (Chapter 11). Characters reserved for delimiting the conditional must be backslash-escaped (spaces, TABs, wildcards and the delimiting ‘‘[’’ and ‘‘]’’).

Mapping Conditionals

Conditional	Description
ac:	browser-accepted content types (‘‘Accept:’’ request header field)
al:	browser-accepted languages (‘‘Accept-Language:’’ request header field)
as:	browser-accepted character sets (‘‘Accept-Charset:’’ request header field)
ck:	cookie data (‘‘Cookie:’’ request header field)
ex:	extended file specification (boolean)
fo:	request forwarded by proxy/gateway host(s) (‘‘Forwarded:’’ request header field)
ho:	browser host internet name or address
hm:	browser host internet address compare to dotted-decimal and mask
me:	request HTTP method
mp:	derived map path (after SCRIPT or MAP rule)
no:	‘‘notepad’’ contents
pa:	first or second pass (after script resolution), as ’1’ or ’2’
pi:	path information
qs:	query string
rc:	internally redirected count, as ’0’, ’1’, ’2’ ..

Conditional	Description
rf:	refering page (‘‘Referer:’’ request header field)
ru:	request URI (non-URL-decoded path)
sc:	request *scheme* (protocol), ‘‘http’’, and if SSL is in use ‘‘https’’ (Chapter 17)
sn:	server name
sp:	server port
st:	script name (after first pass script resolution)
ua:	browser (‘‘User-Agent:’’ request header field)
vs:	virtual host and port request directed to (‘‘Host:’’ request header field)
xf:	proxied client (‘‘X-Forwarded-For:’’ request header field)

Examples

Note

It is possible to *spoof* (impersonate) internet host addresses. Therefore any controls applied using host name/address information cannot be used for authorization purposes in the strictest sense of the term.

1. The following example shows a rule being applied only if the client host is within a particular subnet. This is being used to provide a ‘‘private’’ home page to those in the subnet while others get a ‘‘public’’ page by the second rule.

```
pass / /web/internal/ [ho:131.185.250.*]
pass / /web/
```

2. This is a similar example to the above, but showing multiple host specifications and specifically excluding one particular host using the negation operator ‘‘!’’. This could be read as *pass if ((host OR host) AND (not host))*.

```
pass / /web/internal/ [ho:*.fred.com ho:*.george.com] [!ho:you.fred.com]
pass / /web/
```

3. The next example shows how to prevent browsing of a particular tree except from specified host addresses.

```
pass /web/internal/* /web/SorryNoAccess.html [!ho:131.185.250.*]
pass /web/internal/*
```

   This could be used to prevent browsing of the server configuration files (an alternative to this sort of approach is to use the authorization file, see Chapter 15).

```
pass /httpd/-/* /web/SorryNoAccess.html [!ho:131.185.250.201]
```

4. This example performs much the same task as the previous one, but uses whole conditional negation to prevent browsing of a particular tree except from specified addresses (as well as using the continuation character to provide a more easily comprehended layout . . . note the trailing spaces as required). This could be read as *pass if not (host OR host OR host)*.

```
pass /web/internal/* /web/SorryNoAccess.html \
![\
ho:131.185.250.* \
ho:131.185.251.* \
ho:131.185.45.1 \
ho:ws2.wasd.dsto.gov.au\
]
pass /web/internal/*
```

5. This example demonstrates mapping pages according to geography or language preference (it's a bit contrived, but . . .)

```
pass /doc/* /web/doc/french/* [ho:*.fr al:fr]
pass /doc/* /web/doc/swedish/* [ho:*.se al:se]
pass /doc/* /web/doc/english/*
```

6. How to exclude specific browsers from your site (how many times have we seen this!)

```
# I had to pick on a well-known acronym, no offence Bill!
pass /* /web/NoThankYou.html [ua:*MSIE*]
```

7. This example allows excluding certain requests from specific addresses. This could be read as *pass if ((method is POST) AND (not host))*.

```
pass /* /web/NotAllowed.html [me:POST] [!ho:*.my.net]
```

8. The following illustrates using the server name and/or server port to conditionally map servers executing on clustered nodes using the same configuration file, or for multi-homed/multi-ported hosts. Distinct home pages are maintained for each system, and on BETA two servers execute, one on port 8000 that may only be used by those within the specified network address range.

```
pass / /web/welcome_to_Alpha.html [sn:alpha.*]
pass / /web/welcome_to_Beta.html [sn:beta.*] [sp:80]
pass /* /sorry_no_access.html [sn:beta.*] [sp:8000] [!ho:*.my.sub.net]
pass / /web/welcome_to_Beta_private.html [sn:beta.*] [sp:8000]
```

9. Each of these three do basically the same thing, just using the three possible delimiters according to the characters required in the message. The server generates a 403 response with has as it's text the following message.

```
pass /private/* "403 Can't go in there!" [!ho:my.host.name]
pass /private/* '403 "/private/" is off-limits!' [!ho:my.host.name]
pass /private/* {403 Can't go into "/private/"} [!ho:my.host.name]
```

10. This example illustrates the use of a host network mask, the ''HM:'' conditional.

```
pass /private/* "403 Can't go in there!" [!hm:131.185.250.128/255.255.255.192]
```

The mask is a dotted-decimal network address, a slash, then a dotted-decimal mask. This example shows a 6 bit subnet. Network mask conditionals operate 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. Using the above example the host 131.185.250.250 would be accepted, but 131.185.250.50 would be rejected.

Note that rule processing for any particular path may be checked using the WATCH facility from the Server Administration page. See Chapter 19 for details.

13.8 Mapping User Directories (*tilde* character (“~”))

The convention for specifying user web areas is ‘‘/~username/’’. The basic idea is that the user’s web-available file-space is mapped into the request in place of the tilde and username.

13.8.1 Using The SYSUAF

The USER rule maps a VMS user account default device and directory (i.e. *home* directory) into a request path (Section 13.4.3). That is, the base location for the request is obtained from the VMS systems SYSUAF file. A user’s home directory information is cached, to reduce load on the authorization databases. As this information is usually quite static there is no timeout period on such information (although it may be flushed to make room for other user’s). Cache contents is include in the Mapping Rules Report (Section 18.4) and is implicitly flushed when the server’s rules are reloaded (Section 18.6).

The following is a typical usage of the rule.

```
USER  /~*/*  /*/www/*
```

Note the ‘‘/www’’ subdirectory component. It is **stongly recommended** that users never be mapped into their top-level, but into a web-specific subdirectory. This effectively ‘‘sandboxes’’ Web access to that subdirectory hierarchy, allowing the user privacy elsewhere in the home area.

To accomodate request user paths that do not incorporate a trailing delimiter after the username the following redirect may be used to cause the browser to re-request with a more appropriate path (make sure it follows the USER rule).

```
REDIRECT  /~*  ///~*/
```

WASD also ‘‘reverse maps’’ VMS specifications into paths and so requires additional rules to provide these mappings. (Reverse mapping is required during directory listings and error reporting.) For the continuing example the following rules would be required (and in the stated order).

```
USER  /~*/*  /*/www/*
REDIRECT  /~*  ///~*/
PASS  /~*/*  /user$disk/*/www/*
```

Where user home directories are spread over multiple devices (physical or concealed logical) a reverse-mapping rule would be required for each. Consider the following situation, where user directories are distributed across these devices (concealed logicals)

```
USER$GROUP1:
USER$GROUP2:
USER$GROUP2:
USER$OTHER:
```

This would require the following mapping rules (in the stated order).

```
USER  /~*/*  /*/www/
PASS  /~*/*  /user$group1/*/www/*
PASS  /~*/*  /user$group2/*/www/*
PASS  /~*/*  /user$group3/*/www/*
PASS  /~*/*  /user$other/*/www/*
```

Accounts with a search list as a default device (e.g. SYS$SYSROOT) present particular complications in this schema and should be avoided.

Note

Accounts that possess SYSPRV, are CAPTIVE, have been DISUSERED or that have expired passwords will not be mapped. A ‘‘directory not found’’ error report is returned. This error was chosen to make it to make more difficult to *probe* the authorization environment, determining whether accounts exist or not.

Of course vanilla mapping rules may be used to provide for special cases. For instance, if there is requirement for a particular, privileged account to have a user mapping that could be provided as in the following (rather exagerated) example.

```
PASS  /~system/*  /sys$common/sysmgr/www/*
USER  /~*/*  /*/www/
PASS  /~*/*  /user$disk/*/www/*
```

User Account Scripting

In some situations it may be desirable to allow the average Web user to experiment with or implement scripts. With WASD 7.1 and later, and VMS V6.2 and later, this is possible. Detached scripting must be enabled, the /PERSONA startup qualifier used, and appropriate mapping rules in place. If the SET ‘‘script=as=’’ mapping rule specifies a tilde character then for a user request the mapped SYSUAF username is substituted.

The following example shows the essentials of setting up a user environment where access to a subdirectory in the user’s home directory, [.WWW] with script’s located in a subdirectory of that, [.WWW.CGI-BIN].

```
SET   /~*/www/cgi-bin/*  script=as=~
UXEC  /~*/cgi-bin/*  /*/www/cgi-bin/*
USER  /~*/*  /*/www/*
REDIRECT  /~*  /~*/
PASS  /~*/*  /dka0/users/*/*
```

For more detailed information see the ‘‘Scripting Overview, Introduction’’.

13.8.2 Without Using The SYSUAF

Deprecated and Discouraged

There are now ‘‘better’’ approaches to achieving the same functionality as described in this section. This documentation is retained only for reference by older site configurations.

The server is also able to map user directories using the same mechanisms as for any other. No reference needs to be made to the SYSUAF, user support can be accomplished via a combination of mapping rule and logical name. This approach relies on a correspondance between the username and the home directory name. Hence users are made known by the HTTPd using the name of their top-level directory. User scripts can also be supported using WASD’s DECnet scripting environment.

The ‘‘PASS’’ rule provides a wildcard representation of users’ directory paths. As part of this mapping a subdirectory specifically for the hypertext data should always be included. **Never** map users’ top-level directories. For instance if a user’s account home directory was located in the area USER$DISK:[DANIEL] the following rule would potentially allow the user DANIEL to provide web documents from the home subdirectory [.WWW] (if the user has created it) using the accompanying URL:

```
pass /~*/* /user$disk/*/www/*
```

```
http://host/~daniel/
```

It is **recommended** that a separate logical name be created for locating user directories. This helps hide the internal organisation of the file system. The following logical name definition and mapping rule illustrate this point.

```
$ DEFINE /SYSTEM /EXEC /TRANSLATION=CONCEALED WWW_USER device:[USER.]
```

```
pass /~*/* /www_user/*/www/*
```

Where users are grouped into different areas of the file system a logical search list may be defined.

```
$ DEFINE /SYSTEM /EXEC /TRANSLATION=CONCEALED -
         WWW_USER -
         DISK1:[GROUP1.], -
         DISK1:[GROUP2.], -
         DISK2:[GROUP3.], -
         DISK2:[GROUP4.]
```

```
pass /~*/* /www_user/*/www/*
```

As logical search lists have specific uses and some complications (e.g. when creating files) this is the only use for them recommended with this server, although it is specifically coded to allow for search lists in document specifications.

If only a subset of all users are to be provided with WWW publishing access either their account directories can be individually mapped (best used only with a small number) or a separate area of the file system be provided for this purpose and specifically mapped as user space.

Of course, user mapping is amenable to all other rule processing so it is a simple matter to redirect or otherwise process user paths. For instance, the published username does not need to, or need to continue to, correspond to any real user area, or the user’s actual name or home area:

```
redirect /~doej/* http://a.nother.host/~doej/*
 pass /~doej/* /www/messages/deceased.html
 pass /~danielm/* /special$www$area/danielm/*
 pass /~Mark.Daniel/* /user$disk/danielm/www/*
 pass /~*/* /www_user/*/www/*
```

A user directory is always presented as a top-level directory (i.e. no parent directory is shown), although any subdirectory tree is accesssable by default.

Chapter 14
Authorization Quick Guide

WASD offers a comprehensive and versatile authentication and authorization environment. A little too comprehensive, often leaving the new administrator wondering where to begin. The role of this chapter is to provide a starting place, especially for sources of authentication, along with some basic configurations. Chapter 15 contains a detailed explanation of all aspects. All examples here assume a standard installation and environment.

Just to clarify. **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 that identity.

14.1 SYSUAF/Identifier Authentication

This setup allows any active account to authenticate using the local VMS username and password. By default not every account may authenticate this way, only those holding specified VMS rights identifiers. See Section 15.10.2. The examples provided in this section allows access to the WASD online Server Administration facility, and so may be followed specifically for that purpose, as well as serve as a general guide.

- Define the following logical before calling the server startup procedure. To make such a definition permanent add it to the system or Web environment startup procedures. This logical contains a startup qualifier that configures the server to allow authentication from the SYSUAF, using VMS rights identifiers (Section 15.2).

```
$ DEFINE /SYSTEM HTTPD$STARTUP_SERVER "/SYSUAF=ID"
$ @device:[HT_ROOT.LOCAL]STARTUP.COM
```

- Decide on an identifier name. This can be an existing identifier, or one created for the purpose. For this example the identifier will be ''WASD_WEBADMIN''. Any identifier can be created using actions similar to the following example.

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD /IDENTIFIER WASD_WEBADMIN
```

- Modify the authorization configuration file, accessed by the server using the system logical HTTPD$AUTH, to contain the following. This allows full access to the online Server Administration facility and [.LOCAL] directory (and no world access). Additional paths may be added as required, and of course multiple identifiers may be created and used for multiple realms and paths.

```
["Web Admin"=WASD_WEBADMIN=id]
/httpd/-/admin/* r+w
/ht_root/local/* r+w
```

- The identifier must then be granted to those accounts allowed to authenticate in this way.

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> GRANT /IDENTIFIER WASD_WEBADMIN SYSTEM
```

- Using this approach useful discrimination may be exercised. For instance, one identifier for Web administrators, another (or others) for different authentication requirements.

```
["Web Admin"=WASD_WEBADMIN=id]
/ht_root/local/* r+w
/httpd/-/admin/* r+w
["Area Access"=area-identifier-name=id]
/web/area/* r+w ; r
```

  Of course the one account may hold multiple identifiers and so may have access to various areas.

```
UAF> GRANT /IDENTIFIER WASD_WEBADMIN SYSTEM
 UAF> GRANT /IDENTIFIER area-identifier-name SYSTEM
```

  Using VMS rights identifiers allows significant granularity in providing access.

After Changes

If the HTTPD$AUTH configuration file is changed, or rights identifiers are granted or revoked from accounts, the server should be directed to reload the file and purge any cached authorization information.

```
$ HTTPD/DO=AUTH=LOAD
$ HTTPD/DO=AUTH=PURGE
```

14.2 SYSUAF Authentication

The use of rights identifiers (described above) is strongly recommended. However if this is impractical for some reason, an alternative setup allows any active, non-privileged account to authenticate using the server system VMS username and password.

- Define the following logical before calling the server startup procedure. This logical configures the server to allow authentication from the SYSUAF (Section 15.2).

```
$ DEFINE /SYSTEM HTTPD$STARTUP_SERVER "/SYSUAF"
$ @device:[HT_ROOT.LOCAL]STARTUP.COM
```

- Modify the authorization configuration file, accessed by the server using the system logical HTTPD$AUTH, to contain the following. This allows full access to the Server Administration facility and [.LOCAL] directory (and no world access at all). Additional paths may be configured as required.

```
["Whatever you want to call it!"=VMS]
/web/area/* r+w
```

  Update access (read+write) may be provided to the authorized and read-only access to the world using syntax similar to the following example. This applies to all authentication sources.

```
/web/area/* r+w ; r
```

- With no discrimination on which accounts can and cannot authenticate using this setup additional controls on what can be accessed must be supplied. This is done using either restriction lists (Access Restriction List).

```
["Realm Name"=VMS]
/web/area1/* r+w,~username1,~username2
/web/area2/* r+w,#host1,#host2,~username3
```

  or using access groups (Section 14.4)

```
["Realm Name"=VMS;list_name1=list]
/web/area1/* r+w ;
["Realm Name"=VMS;list_name2=list]
/web/area2/* r+w ; r
```

Note

To also allow privileged accounts to authenticate via the SYSUAF with this configuration change the logical definition as follows.

```
$ DEFINE /SYSTEM HTTPD$STARTUP_SERVER "/SYSUAF=RELAXED"
```

This is not a recommended configuration.
Using VMS identifiers is a far better solution. See previous section.

14.3 Other Authentication

Other sources of authentication are available, either by themselves or used in the same configuration file (different realms and paths) as those already discussed (Section 15.5). Non-SYSUAF sources do not require any startup qualifier to be enabled.

- **ACME** DOIs (Authentication and Credential Management Extension, Domains of Interpretation) may be used to authenticate requests.

```
["Whatever you want to call it!"=doi=ACME]
/web/area/* r+w
```

- **Simple lists** contain usernames and unencrypted passwords. These are plain-text files, created and modified using any desired editor.

```
["Whatever you want to call it!"=list-name=list]
/web/area/* r+w
```

This is a very simple arrangement, with little inherent security. Lists are more useful when grouping names together for specifying which group may do what to where.

- **HTA databases** are WASD-specific, binary repositories of usernames, encrypted passwords, capabilities, user and other detail.

```
["Whatever you want to call it!"=HTA-database-name=HTA]
/web/area/* r+w
```

  These databases may be administered using the online Server Administration facility (Section 18.5) or the HTAdmin command-line utility (Section 23.7), are quite secure and versatile.

- **External agents** are authentication and authorization scripts executed on demand, under the control-of but external to the server. It is possible for a site to write it's own, custom authorization agent.

```
["Whatever you want to call it!"=agent-name=agent]
/web/area/* r+w
```

- **X.509** establishes identity based on Public Key Infrastructure (PKI) authentication certificates. This is only available for SSL transactions.

```
[X509]
/web/area/* r+w
```

- **RFC1413** IETF document describes an identification protocol that can be used as a form of *authentication* within this realm.

```
["Whatever you want to call it!"=RFC1413;A_PROJECT=list]
/web/area/* r+w ; r
```

14.4 Read and Write Groupings

WASD allows separate sources for groups of usernames to control read and write access in a particular realm (Section 15.6). These groups may be provided via simple lists, VMS identifiers, HTA databases and authorization agents. The following example shows an identifier authenticated realm with full and read-only access controlled by two simple lists. For the first path the world has no access, for the second read-only access (with the read-only grouping becoming basically redundant information).

```
["Realm Name"=identifier_name=id;full_access_name=list;read-only_name=list]
/web/area/* r+w ;
/web/another-area/* r+w ; r
```

14.5 Considerations

Multiple authentication sources (realms) may be configured in the one HTTPD$AUTH file.

Multiple paths may be mapped against a single authentication source.

Any path may be mapped only once (for any single virtual service).

Paths may have additional access restrictions placed on them, including client host name, username, etc (Access Restriction List).

The configuration file is loaded and stored by the server at startup. If changed it must be reloaded to take effect. This can be done manually using

```
$ HTTPD/DO=AUTH=LOAD
```

Authentication information is cached. Access subsequently removed or modified will not take effect until the entry expires, or is manually purged using

```
$ HTTPD/DO=AUTH=PURGE
```

Failed attempts to authenticate against a particular source are limited. When this is exceeded access is always denied. If this has happened the cache must be manually purged before a user can successfully authenticate

```
$ HTTPD/DO=AUTH=PURGE
```

Chapter 15
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.

Authentication environments can get complex very quickly, don't forget to ''keep it simple, stupid'', see Section 15.8.1.

Overview

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 Section 13.4.5.

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 refers to a collection of usernames and associated *permissions*.

15.1 Rule Interpretation

The configuration file rules are scanned from first towards last, until a matching rule is encountered. 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 (Chapter 12), 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 (Chapter 11).

15.2 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 17), thereby ensuring that the authorization environment is not compromised by use in non-encrypted transactions. Two HTTPd 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 15.13).
  - **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 15.10.2).
  - **PROXY** allows non-SYSUAF to SYSUAF username proxying (Section 15.10.4).
  - **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 15.10.3.
  - **(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 Section 13.7 and Access Restriction List.

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 [AuthFailurePeriodSeconds] period puts it into break-in evasion mode after which the period [AuthFailureTimeoutSeconds] 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.
- **[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.

Authentication Failures

Details of authentication failures are logged to the HTTPd 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 (Section 6.10).

15.3 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	PUT
READ or R	no	yes	yes	no	no
WRITE or W	yes	no	no	yes	yes
R+W	yes	yes	yes	yes	yes
NONE	no	no	no	no	no
DELETE	yes	yes	no	no	no
GET	no	yes	no	no	no
HEAD	no	no	yes	no	no
POST	no	no	no	yes	no
PUT	no	yes	no	no	yes

15.4 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 HTTPD$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 Section 6.12.

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 ''[Auth-Proxy]'' 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 15.5.

- **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 15.10.4).

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 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.
- **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 15.13).
- **PROMISCUOUS -** This realm is only available while the /PROMISCUOUS qualifier is in use (Chapter 18).
- **RFC1413 -** This IETF document describes an identification protocol that can be used as a form of *authentication* within this realm.
- **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 17) 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 15.5.

  See Section 15.10 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 17 for further information on SSL, and Section 17.3.7 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 List

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.

- **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).

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.

15.5 Authorization Sources

Username authorization information may be derived from 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 15.10.2.

  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 15.5). Further information on SYSUAF authentication may be found in Section 15.10. These examples illustrate the general idea.

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

- **ACME**

  The Authentication and Credential Management Extension (ACME) can be used to authenticate requests on Alpha (and IA64?) running VMS V7.3 or later. Two ACME agents are currently available (as at WASD v8.5), "VMS" (SYSUAF) and "MSV1_0" (Microsoft domain authentication used by Advanced Server). Others, including Kerberos and LDAP, have been suggested as candidates for development and future release. There is a 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. The ACME service should also be used to process WASD SYSUAF authentication where possible (Section 15.10.1).

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 HT_AUTH: directory, have an extension of ‘‘.$HTL’’, and do not need to be world accessable.

```
# 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 18.5) or the HTAdmin command-line utility (Section 23.7). They are located in the HT_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 17 for further information on SSL, and Section 17.3.7 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 it's own internal authentication/authorization processing, and using the script *callout* mechanism returns it's 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 Scripting Environment’’ document for information on the use of callouts, and the example and working authorization agents provided in the HT_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=HT_ROOT:[LOCAL]CEL1.LIS
/a/nother/path/* r+w,param=HT_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 leading parameter of ‘‘/NO401’’.

```
["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 HTTPD$MAP configuration file:

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

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

- **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 15.4) 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 List) 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.

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 15.10.3.

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.

15.6 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 15.3.

- **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.

15.7 Virtual Servers

As described in Section 6.3, 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.wasd.dsto.defence.gov.au: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.wasd.dsto.defence.gov.au:80]]
# BETA has it's own HTA database
[BETA_USER=hta]
/web/* r+w ; r
[[gamma.wasd.dsto.defence.gov.au: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 18.4) 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.

15.8 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; *.dsto.defence.gov.au,get
/web/socialclub/accounts/* 131.185.45.*,~BLOGGS,get,post; *.dsto.defence.gov.au,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. 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
```

5. 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
```

6. 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
```

7. 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
```

8. 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
```

9. 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
```

15.8.1 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
/ht_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

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

15.9 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 5.5.2) or via the Server Administration facility (Chapter 18). Of course the other solution is just to disable caching, which is a less than optimal solution.

15.10 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 (''http:'') authentication is transmitted encoded but not encrypted, making it 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 15.10.2 immediately below. Accounts that are disusered, have passwords that have expired or that are captive or restricted are always rejected. Accounts that have access time restricting REMOTE or NETWORK access will have those restrictions honoured (see Section 15.10.2 for a workaround for this).

Also see Section 15.10.5.

15.10.1 ACME

The Authentication and Credential Management Extension (ACME) can be used to authenticate SYSUAF requests on Alpha (and IA64?) running VMS V7.3 or later (Section 15.5). WASD allows the ACME facility to be substituted for it's own SYSUAF authentication routines. The immediate advantage of using ACME is 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.

To enable ACME authentication for the WASD "VMS" realm add the following configuration directive to HTTPD$CONFIG. No other changes are required.

```
[AuthSYSUAFuseACME]  enabled
```

15.10.2 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 15.10.5.
- **WASD_PASSWORD_CHANGE -** allows an account to modify it's SYSUAF password, if this is configured for the server, see Section 15.14.
- **WASD_PROXY_ACCESS -** allows an account to be used for proxy access if /SYSUAF=ID is in effect, see Section 15.10.4.

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 15.9).

15.10.3 WASD "Hard-Wired" Identifiers

Deprecated and Discouraged

There are now ''better'' approaches to achieving the same functionality as described in this section. This documentation is retained only for reference by older site configurations.

This description is included for reasons of backward compatibility. As of version 6.0 the WASD identifiers must be enabled using the /SYSUAF=WASD qualifier.

When SYSUAF authentication is enabled, by default all non-privileged, active accounts are capable of authentication. Restriction of this to those actually requiring such a capability is provided using VMS rights identifiers. When the /SYSUAF=WASD qualifier is employed a VMS account must possess one of two specific identifiers before it is allowed to be used for server authentication. Note that this mechanism can also allow privileged accounts to be so used . . . deploy with discretion! Note also that the use of access identifiers with this facility allows a much finer control of which accounts may be used for authentication and so is also the prefered mechanism for deploying SYSUAF authentication.

1. **WASD_VMS_R**

   This identifier provides at most read access.

2. **WASD_VMS_RW**

   This identifier provides read and write access (path protections still apply of course).

Other identifiers provide further control for the way in which the authenticated account may be used.

1. **WASD_VMS_HTTPS**

   Use of the authenticated account is restricted to ''https:'' (SSL) requests.

2. **WASD_VMS_PWD**

   The account is allowed to change it's SYSUAF password (!!). It is recommended this facility only be employed with SSL in place.

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

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

   The authorization configuration file must provide authenticated access.

   ```
   [VMS;VMS]
   /httpd/-/change/vms/* https:,r+w
   ```

   Also see Section 15.14.

3. **WASD_VMS__<group-name>**

This form allows a suitably named identifier to be created for use in providing group-membership via the SYSUAF. Note the double-underscore separating the fixed from the locally specified portion. Using these identifiers it is possible to limit paths to SYSUAF-authenticated accounts possessing the requisite identifier in manner similar to non-SYSUAF-authentication groups. An account possessing the WASD_VMS_ _TESTING identifier is allowed write access to the path in the following example:

```
[VMS;TESTING]
/web/project/testing/* r+w ; r
```

All four rights identifiers must exist for the /SYSUAF=ID facility to be used (even though none may be granted to any account). The identifiers may be created using the AUTHORIZE utility with following commands:

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD /IDENTIFIER WASD_VMS_R
UAF> ADD /IDENTIFIER WASD_VMS_RW
UAF> ADD /IDENTIFIER WASD_VMS_HTTPS
UAF> ADD /IDENTIFIER WASD_VMS_PWD
```

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

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

and removed using:

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

15.10.4 VMS Account Proxying

Any authentication realm can have it's 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 15.10.2 (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 it's 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]  HT_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.

15.10.5 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 15.2). 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 15.10.2), 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 15.10.2 above).

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

15.10.6 SYSUAF and SSL

When SSL is in use (Chapter 17) 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 HTTPD$MAP configuration file:

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

or alternatively the following to the HTTPD$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 15.2.

15.10.7 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 15.9). 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 18.6).

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 applies to all SYSUAF authenticated paths, but may be disabled and enabled on a per-path basis using the SET *noprofile* and *profile* mapping rules (Section 13.4.5).

```
# HTTPD$MAP
set /ht_root/local/* profile auth=all
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 HT_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.

15.10.8 SYSUAF Profile For Full Site Access

Much of a site's package directory tree is inaccessable 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 15.10.7), 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 /$ht_root/ path (with the inline ''profile'').

```
# HTTPD$MAP
set * noprofile
  .
  .
  .
pass /ht_root/* /ht_root/*
pass /$ht_root/* /ht_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.

```
# HTTPD$AUTH
[["/$ht_root/ Access"=WASD_TREE_ACCESS=id]]
/$ht_root/* https,10.1.1.2,~system,read
```

15.11 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's a method for allowing ad hoc authenticated access to the server, primarily intended for non-configured access to the online Server Administration facilities (Section 18.1) 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
```

15.12 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, HT_ROOT:[EXAMPLE]AUTHACE.COM.

15.13 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 HT_ROOT tree
/ht_root/*
```

There is also a per-path equivalent of the /AUTHORIZE=ALL functionality, described in Section 13.4.5. 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
```

15.14 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. Note that 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). For example:

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

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

Use some form of cautionary wrapper if providing this functionality:

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

<BLOCKQUOTE><I>
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.
</I></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. Expired passwords may be specially processed by specifying a URL with [AuthSysUafPwdExpURL] configuration directive (Section 8.2). 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 HTTPD$CONFIG directive

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

and the HTTPD$AUTH rule

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

would allow expired passwords to be changed.

The HTTPD$MAP *set auth=sysuaf=pwdexpurl=<string>* rule allows the same URL to be specified on a per-path basis.

15.15 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 its 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 it's authorization cache, effectively requiring new authorization information the next time that path is accessed.

The basic procedure is as follows.

1. Add the query string ''?httpd=cancel'' to the path in question (if there is an existing query then replace it), as in the following example.

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

2. The browser will respond with an authorization failure, and prompting to retry or reenter the username and password.

3. It is necessary to clear at least the password (i.e. remove any password from the appropriate field) and reenter.

4. The browser again responds with an authorization failure.

5. At this stage the authorization dialog can be cancelled, resulting in a server authorization failure message.
6. 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.

Chapter 16
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. The following steps provide a brief outline of proxy configuration.

1. Enable proxy serving and specify which particular services are to be proxies (Section 16.1.1 and Chapter 9).
2. If proxy caching is required (most probably, see Section 16.2)
   - Decide on a cache device, create the cache root directory, modify server startup procedures to include the HT_CACHE_ROOT logical name (Section 16.2.1).
   - Enable caching on required services (Section 16.2.2).
   - Adjust relevant cache management configuration parameters if required (Section 16.2.3).

- If required adjust cache retention parameter (Section 16.2.5).

3. If providing SSL tunnelling (proxy of Secure Sockets Layer transactions) add/modify a service for that (Section 16.3).
4. Add HTTPD$MAP mapping rules for controlling this/these services (Section 16.1.4, Section 16.3.2, and Section 16.4).
5. Restart server (HTTPD/DO=RESTART).

Error Messages

When proxy processing is enabled and HTTPD$CONFIG directive [ReportBasicOnly] is disabled it is necessary to make adjustments to the contents of the HTTPD$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 accessable.

16.1 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.

16.1.1 Enabling A Proxy Service

Proxy serving is enabled on a per-server basis using the [ProxyServing] configuration parameter.

WASD can configure services using the HTTPD$CONFIG [service] directive, the HTTPD$SERVICE configuration file, or even the /SERVICE= qualifier.

HTTPD$CONFIG [Service]

The actual services providing the proxy serving (i.e. the host and port) are specified on a per-service basis. This means it is possible to have proxy and non-proxy services deployed on the one server (on different ports of course). Proxying is enabled by appending the *proxy* keyword to the particular service specification. The following example shows a non-proxy and proxy service.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:80
http://alpha.wasd.dsto.defence.gov.au:8080;proxy
```

HTTPD$SERVICE

Proxy service configuration using the HTTPD$SERVICE configuration is slightly simpler, with a specific configuration directive for each aspect. (Chapter 9). This example illustrates configuring the same services as used in the previous section.

```
[[http://alpha.wasd.dsto.defence.gov.au:80]]

[[http://alpha.wasd.dsto.defence.gov.au:8080]]
[ServiceProxy]  enabled
```

Examples in following section all show configuration using the HTTPD$CONFIG [Service] directive. When using the HTTPD$SERVICE configuration file Server Administration facility interface all relevant proxy directives are provided for selection.

16.1.2 Proxy Bind

Using the HTTPD$MAP SET *proxy=bind=<IP-address>* rule it becomes possible to make the outgoing request appear to originate from a particular source. The Network Interface must be able to bind to the specified IP address (i.e. it cannot be an arbitrary address).

```
SET http://*.fred.com proxy=bind=131.185.250.1
```

16.1.3 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 it's proxied requests via the next proxy server in the hierarchy. This is known as *proxy chaining*. Using the *chain* keyword specify the host name of the next server when enabling the proxy service, as in this example:

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:8080;proxy;chain=next.proxy.host
```

Chaining may also be controlled on a virtual service or path basis using the HTTPD$MAP SET *proxy=chain=<host:port>* rule.

```
SET http://*.com proxy=chain=next.proxy.host:8080
```

16.1.4 Controlling Proxy Serving

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.

The following example shows a service specification using the ''pauth'' parameter making the proxy service require authorization for use.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:8080;proxy;pauth
```

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 HTTPD$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.wasd.dsto.defence.gov.au:8080]]
["Proxy Access"=PROXY_ACCESS=id]
http://* read+write
```

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, only using the ''lauth'' parameter.

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 (Chapter 13 and Section 13.7) possibly in the context of a virtual service specification for the particular connect service host and port (Section 13.6). 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.wasd.dsto.defence.gov.au: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.wasd.dsto.defence.gov.au: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.wasd.dsto.defence.gov.au: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.wasd.dsto.defence.gov.au: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.wasd.dsto.defence.gov.au: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.wasd.dsto.defence.gov.au: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.

16.2 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 it's 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 it's 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

Proxied *responses* will only be cached if . . .

- status code begins with ''2'' (success)
- 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 a *Pragma: no-cache* field
- does not exceed a configuration parameter in size

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 16.2.6 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.

16.2.1 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 HT_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 (STARTUP.COM) 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 [HT_CACHE] and may be created manually using the following command.

```
$ CREATE /DIR /OWN=HTTP$SERVER /PROT=(O:RWED,G,W) device:[HT_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.

HT_CACHE_ROOT Logical

It is required to define the logical name HT_CACHE_ROOT if any proxy services are specified 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 HT_CACHE_ROOT device:[HT_CACHE.]
```

If example startup procedure is in use then it is quite straight-forward to have the logical created during server startup (STARTUP.COM).

16.2.2 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.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:80
http://alpha.wasd.dsto.defence.gov.au:8080;proxy;cache
```

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.wasd.dsto.defence.gov.au: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

```
[VMS]
/ht_cache_root/* ~webadmin,131.185.250.*,r+w ;
```

16.2.3 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 it's 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 it's 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 it's 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
```

16.2.4 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 it's ''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 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 16.2.5.
- 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).

16.2.5 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 it's 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.

16.2.6 Reporting and Maintenance

The HTTPDMON utility allows real-time monitoring of proxy serving activity (Section 23.8).

Proxy reports and some administrative control may be exercised from the online Server Administration facility (Chapter 18). 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 HTTPD$CONFIG 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.

16.2.7 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 HT_ROOT:[SRC.UTILS]PCACHE.C program provides a versatile command-line utility as well as CGI(plus) script, making cache file information accessable 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 HT_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 HTTPD$MAP configuration file.

```
pass /ht_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

```
[VMS]
/pcache/* ~webadmin,131.185.250.*,r+w ;
/ht_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.

16.3 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.

The WASD CONNECT service implements the de facto standard HTTP **CONNECT** method, described in a number of Internet Drafts.

16.3.1 Enabling CONNECT Serving

As with proxy serving in general, CONNECT serving may enabled on a per-service basis using the HTTPD$CONFIG [service] directive, the HTTPD$SERVICE configuration file, or even the /SERVICE= qualifier.

HTTPD$CONFIG [Service]

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.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:80
http://alpha.wasd.dsto.defence.gov.au:8080;proxy
http://alpha.wasd.dsto.defence.gov.au:8081;connect
http://alpha.wasd.dsto.defence.gov.au:8082;proxy;connect
```

HTTPD$SERVICE

Proxy service configuration using the HTTPD$SERVICE configuration is slightly simpler, with a specific configuration directive for each aspect (Chapter 9). This example illustrates configuring the same services as used in the previous section.

```
[[http://alpha.wasd.dsto.defence.gov.au:80]]

[[http://alpha.wasd.dsto.defence.gov.au:8080]]
[ServiceProxy]  enabled

[[http://alpha.wasd.dsto.defence.gov.au:8081]]
[ServiceProxySSL]  enabled

[[http://alpha.wasd.dsto.defence.gov.au:8082]]
[ServiceProxy]  enabled
[ServiceProxySSL]  enabled
```

16.3.2 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 mapping rules, as in the following example:

```
[[alpha.wasd.dsto.defence.gov.au:8080]]
pass *:443 [me:connect]
pass * "403 CONNECT only allowed to port 443." [me:connect]
```

All of the comments on the use of general and conditional mapping made in Section 16.1.4 can also be applied to the connect service.

16.4 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’s 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.

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 (Section 8.2), or the ‘‘#!+’’ MIME.TYPES directive (Section 6.6.2).

Rules required in HTTPD$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 /*/-/* /ht_root/runtime/*/*
```

16.4.1 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".

Keyword	Description
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.

16.4.2 "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.

16.5 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 it's behalf.

Request Redirect

The basic mechanism allowing this gatewaying is ‘‘internal’’ redirection. The *redirect* mapping rule (Section 13.4.2) 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.

16.5.1 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 16.5).

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 (Section 13.4.2).

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 16.1.4.

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 HT_ROOT:[SRC.HTTPD]PROXYVERIFY.C module.

16.5.2 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 inaccessable 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 it’s 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/
```

16.5.3 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
```

16.5.4 Originating SSL

This proxy function allows standard HTTP clients to connect to Secure Sockets Layer (Chapter 17) services. This is very different to the CONNECT service (Section 16.3), 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 17.3.7) 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 (Chapter 9). 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 it's credentials have not been established.

As with other proxy serving, HTTP-to-SSL gatewaying may enabled on a per-service basis using the HTTPD$CONFIG [service] directive, the HTTPD$SERVICE configuration file, or even the /SERVICE= qualifier, although not all options are available unless using HTTPD$SERVICE.

HTTPD$CONFIG [Service]

The actual services providing the SSL gateway (i.e. the host and port) are specified on a per-service basis, enabled by appending the *pclientssl* keyword to the particular service specification. The following example shows such a services.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:8080;proxy;pclientssl
```

HTTPD$SERVICE

With proxy service configuration being done using the HTTPD$SERVICE configuration file (Chapter 9) is is performed with specific directives. This example illustrates configuring the same services as used in the previous section.

```
[[http://alpha.wasd.dsto.defence.gov.au:8080]]
[ServiceProxy]  enabled
[ServiceClientSSL]  enabled
```

16.6 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.

- **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.wasd.dsto.defence.gov.au 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:

  ```
  wasd.dsto.defence.gov.au,dsto.defence.gov.au,defence.gov.au
  ```

- **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 it's 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 ''defence.gov.au'' 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, ".defence.gov.au"))
        return "DIRECT";
     else
        return "PROXY www.wasd.dsto.defence.gov.au: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 HT_ROOT:[LOCAL] and have a file extension of .PAC (which follows Netscape naming convention).

  The following HTTPD$CONFIG 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.wasd.dsto.defence.gov.au:8080]
pass http://* http://*
pass / /ht_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.wasd.dsto.defence.gov.au:8080/
```

A template for a proxy auto-configuration file may be found at HT_ROOT:[EXAMPLE]PROXY_AUTOCONFIG.TXT.

Chapter 17
Secure Sockets Layer

This section is not a tutorial on SSL. It contains only information relating to WASD's use of it. Refer to the listed references, Section 17.6, for further information on SSL technology.

The *Secure Sockets Layer* protocol (SSL) is designed to provide a secure channel between two communicating applications, in the case of HTTP between the browser (client) and the HTTPd (server). It also authenticates server and optionally client identity. 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 effectively protected from unauthorized access while in transit.

SSL functionality is not supplied with the basic WASD package. In part this is due to the relative bulk of this component, but also considers potential patent issues and export restrictions on some cryptography technology in some jurisdictions.

WASD implements SSL using a freely available software toolkit supported by the **OpenSSL Project**, in particular the VMS port, to which Richard Levitte (levitte@lp.se) and Robert Byer (ByerRA@aol.com) have been significant contributors.

OpenSSL is a continuing development of the **SSLeay** toolkit (pronounced ''S-S-L-E-A-Y'', i.e. all letters spelt), authored by Eric Young and Tim Hudson. 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 Alpha, IA64 and (now - late 2003) VAX 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 ;^).

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! ’’

SSL Overhead

SSL **adds a significant overhead** to an HTTP transaction for the following reasons.

- An initial *connection establishment*, where the client and server exchange cryptographic data and authorize the transaction.
- The transaction transfer, where all *application data* must be processed by the CPU into an encrypted stream, and back . . . very expensive processes.

- The encrypted data contains more bytes than the raw data, increasing network transfer time.
- Other miscellaneous SSL handshaking for the life of the transaction.

For these reasons **SSL HTTP is slower and has far greater impact on the server system CPU** than standard HTTP and therefore should only be used when transaction privacy is required, not as a general HTTP service. Also, if a general HTTP and an SSL HTTP service is provided on a multi-processor system, with one or other or both experiencing significant traffic, then the two services should be run in separate processes.

Interoperability

WASD SSL has been used against a wide variety of browsers and certificates. Most combinations work. Some do not, usually related to the level of encryption required by some certificates precluding export-grade browsers from connecting.

To date OpenSSL certificates, and those from Thawte and VeriSign have been deployed on WASD servers.

WASD supports both 40bit, USA ''export-grade'' encryption, as well as 128-bit, **full-strength, USA ''domestic-grade''** encryption. Note that as of early 2000, 128-bit USA domestic-grade encryption has been generally available due to changes in USA Federal Government export restriction policy. Netscape Navigator 4.73 and Microsoft Internet Explorer 5.5 and later now provide this level of encryption as standard.

The ''/SSL'' qualifier controls which version(s) of the SSL protocol the server will support; ''2'', ''3'' or ''23'' (i.e. versions 2 and 3, also the default). Using /NOSSL disables the SSL functionality of an SSL executable. There are also per-service configuration directives for tailoring the protocol version.

HTTP-to-SSL Gateway

The WASD proxy service can provide an HTTP-to-SSL gateway, allowing standard HTTP clients to connect to Secure Sockets Layer services. See Section 16.5.4.

17.1 SSL Functionality Sources

Secure Sockets Layer functionality is easily integrated into WASD and is available from one (or more) of three sources. See Section 17.2 for the basics of installing WASD SSL and Section 17.3 for configuration of various aspects.

1. The **HP SSL (Secure Sockets Layer) for OpenVMS Alpha/IA64/VAX** product

   http://www.openvms.compaq.com/openvms/products/ssl/ssl.html

   This is provided from the directory SYS$COMMON:[SSL] containing shared libraries, executables and templates for certificate management, etc. If this product is installed and started up the WASD installation and update procedures should detect it and provide the option of compiling and/or linking WASD against it's sharable 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, IA64 (Itanium) and VAX for VMS version 6.0 up to current. Obtain these from the same source as the main package.

   WASD SSL installation creates an OpenSSL directory in the source area, HT_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**.

17.2 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 SSL for OpenVMS Alpha/IA64/VAX 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
     ```

   - UNZIP the WASD SSL package.

     ```
     $ SET DEFAULT HT_ROOT:[000000]
     $ UNZIP "-V" device:[dir]archive.ZIP
     ```

2. It is then necessary to build the HTTPd SSL executables. This can be done in either of two ways.

   - During an original INSTALL or subsequent UPDATE of the entire package. As of v8.1 these 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.

     ```
     $ @HT_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.

   ```
   $ @HT_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 HTTPD$CONFIG configuration file to specify an SSL service. For example the following creates both a standard HTTP service on the default port 80 and an SSL service on the default port 443

```
[Service]
the.host.name
https://the.host.name
```

6. Shutdown the server completely, then restart.

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

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

   Standard HTTP

   ```
   http://the.host.name/
   ```

   SSL HTTP

   ```
   https://the.host.name/
   ```

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 17.4. This may then be used by placing it in the appropriate local directory, assigning the WASD_SSL_CERT symbol appropriately before startup.

17.3 SSL Configuration

The example HTTPd 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 HTTPD$CONFIG [service] directive, the HTTPD$SERVICE configuration file, or even the /SERVICE= qualifier.

17.3.1 HTTPD$CONFIG [Service]

The following example illustrates creating two services using the HTTPD$CONFIG [Service] directive; a standard HTTP service on the default port 80, and an SSL service on the default port 443.

```
[Service]
alpha.host.name
https://alpha.host.name
```

The one further requirement of an SSL server is a *certificate*. By default this is located using the HTTPD$SSL_CERT logical name during startup, however if required, each SSL service can have an individual certificate configured against it using the syntax shown in this example.

```
[Service]
alpha.host.name
https://alpha.host.name;cert=ht_root:[local]alpha.pem
https://beta.host.name;cert=ht_root:[local]beta.pem
```

17.3.2 HTTPD$SERVICE

SSL service configuration using the HTTPD$SERVICE configuration is slightly simpler, with a specific configuration directive for each aspect. See Chapter 9. This example illustrates configuring the same services as used in the previous section.

```
[[http://alpha.host.name:80]]

[[https://alpha.host.name:443]]
[ServiceSSLcert]  ht_root:[local]alpha.pem

[[https://beta.host.name:443]]
[ServiceSSLcert]  ht_root:[local]beta.pem
```

17.3.3 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 HTTPD$SSL_CERT logical name during startup, however if required, each SSL service can have an individual certificate configured against it as shown above.

17.3.4 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 is by default located using the HTTPD$SSL_KEY logical, though can be specified on a per-service basis. 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 (Section 23.8), and if any OPCOM category is enabled via an operator message (Section 6.10). 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 5.5.2). 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.

17.3.5 SSL Virtual Services

Multiple virtual SSL services (https:) sharing the same certificate (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:). Services requiring unique certificates can only be configured for the same port number against individual and unique IP addresses (i.e. not against aliases).

This is not a WASD restriction, it applies to all servers for significant technical reasons. Secure Sockets Layer is designed to *wrap* an entire application protocol (in this case HTTP). HTTP virtual services use the ''Host:'' field of the request header to determine which service the client intended to use. This requires the network connection established and at least the request header transfered and processed. For an SSL service establishing the connection requires a complex transaction involving, amongst other things, certificate exchange. Hence, the certificate (and all other SSL parameters) must be determined at the time the server accepts the initial connection request. At that point the only defining characteristics can be IP address and port, and therefore services requiring unique certificates must be unique either by address or port. Services sharing certificates do not have this restriction and so may be configured against host name aliases.

For example, unique certificates for https://www.company1.com:443/ and https://www.company2.com:443/ can be configured only if COMPANY1 and COMPANY2 have unique IP addresses. If COMPANY2 is an host name alias for COMPANY1 they must share the same certificate. During startup service configuration the server checks for such conditions, forces subsequent services to use the same SSL characteristsics as the first configured, and issues a warning about this ''sharing''.

17.3.6 SSL Access Control

When authorization is in place (Chapter 15) 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 17.3.7). SSL may be used as part of the site's access control policy, as whole-of-site, see Section 15.2, or on a per-path basis, see Section 13.7 and Access Restriction List.

17.3.7 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 15 for general information on WASD authorization and Section 15.4 for configuring a X509 realm. Section 17.6 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 it's Certificate Authority to establish it's 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 stored in a cache, reducing subsequent request overheads significantly. Each cache entry 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.

17.3.8 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 HTTPD$AUTH file specifying a path against an [X509] realm causes the server to suspend request processing and renegotiate with the client to supply a certificate. If a suitable certificate is supplied the request authorization continues with normal processing. This obviously incurs an additional network transaction.
- **Optional access control -** once an acceptable certificate is supplied it can be subject to further access control by matching against its contents. The *Issuer* (CA) and the *Subject* (client) *Distinguished Name* (DN) has various components including the name of the organization providing the certificate (e.g. ''VeriSign'', ''Thawte''), location, common name, email address, etc. Those certificates matching or not matching the parameters are allowed or denied access.
- **Certificate verification -** by default supplied certificates have their CA verified by comparing to a list of recognised CA certificates stored in a server configuration file. If the CA component of the client certificate cannot be verified the connection is terminated before the HTTP request can begin. Although this is obviously required behaviour for authentication there may be other circumstances where verification is not required, a certificate content display service for instance. WASD optionally allows non-verified certificates to be used on a per-resource basis.
- **''Fingerprint'' REMOTE_USER -** when a certificate is accepted by the server it generates a unique *fingerprint* of the certificate. By default, this 32 digit hexadecimal number is used by the server as an *effective username*, one that would normally be supplied via a username/password dialog (as an alternative see the section immediately below). This effective username becomes that available via the CGI variable REMOTE_USER. Although a 32 digit number is not particularly site-administrator friendly it is a unique representation (MD5 digest) of the individual certificate and can be used in HTTPD$AUTH access-restriction directives and included in group lists and databases for full WASD authorization control.

- **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).

17.3.9 X509 Configuration

Of course, the WASD SSL 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 following /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 HTTPD$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 2)

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

1. per-service configuration using HTTPD$SERVICE or HTTPD$CONFIG
2. per-server using /SSL=CAFILE=filename
3. per-server using HTTPD$SSL_CAFILE

By Service

To enable client certification for all requests on a per-service basis the following HTTPD$CONFIG directive may be used. A non-default CA verification file can also optionally be supplied.

```
[Service]
https://the.host.name;verify
https://the.host.name;cafile=HT_ROOT:[LOCAL]CA_THE_HOST_NAME.TXT
```

When HTTPD$SERVICE is in use a service-specific directive is provided for both per-service verification and per-service CA file specification (allowing different services to accept a different mix of CAs).

```
[[https://the.host.name:443]]
[ServiceSSLclientVerifyRequired]  enabled
[ServiceSSLclientCAfile]  HT_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 HTTPD$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 (Section 13.7). 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 17.6 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\ HTTPd\ 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 17.3.11), 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 HTTPD$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
/ht_root/local/* ~106C8342890A1703AAA517317B145BF7,r+w

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

17.3.10 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 HTTPD$SSL_CAFILE logical or per-service ‘‘;cafile=’’ or ‘‘[ServiceSSLclientCAfile]’’ directives.

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 ‘‘#’’ or ‘‘!’’ characters. For WASD this would be best stored in the HT_ROOT:[LOCAL] directory, or site equivalent.

An example of how such a file appears is provided below (ellipses inserted to reduce the bulk of example). There is one of these per certificate authority.

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

```
Verisign Class 1 Public Primary Certification Authority
========================================================
MD5 Fingerprint: 97:60:E8:57:5F:D3:50:47:E5:43:0C:94:36:8A:B0:62
PEM Data:
-----BEGIN CERTIFICATE-----
MIICPTCCAaYCEQDNun9W8N/kvFT+IqyzcqpVMA0GCSqGSIb3DQEBAgUAMF8xCzAJ
BgNVBAYTAlVTMRcwFQYDVQQKEw5WZXJpU2lnbiwgSW5jLjE3MDUGA1UECxMuQ2xh
c3MgMSBQdWJsaWMgUHJpbWFyeSBDZXJ0aWZpY2F0aW9uIEF1dGhvcml0eTAeFw05
  . . .
FvjqBUuUfx3CHMjjt/QQQDwTw18fU+hI5Ia0e6E1sHslurjTjqs/OJ0ANACY89Fx
lA==
-----END CERTIFICATE-----
Certificate Ingredients:
    Data:
        Version: 1 (0x0)
        Serial Number:
            cd:ba:7f:56:f0:df:e4:bc:54:fe:22:ac:b3:72:aa:55
        Signature Algorithm: md2WithRSAEncryption
        Issuer: C=US, O=VeriSign, Inc., OU=Class 1 Public Primary
  . . .
        35:b0:7b:25:ba:b8:d3:8e:ab:3f:38:9d:00:34:00:98:f3:d1:
        71:94

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

The WASD SSL package provides an example CA verification file constructed from all the certificates provided in Netscape Navigator CERT7.DB file. This has been generated for and obtained from the Apache *mod_ssl* package, being used for the same purpose with that. The WASD file name is CA-BUNDLE_CRT.TXT and is usually located in HT_ROOT:[LOCAL]. The exact date and *mod_ssl* version it was obtained from can be found in the opening commentary of the file itself. The contents of this file can easily be pared down to the minimum certificates required for any given site. The more certificates in the file the greater the overhead in verifying any given client.

17.3.11 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 HTTPD$MAP mapping rules. See Section 17.5. 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"
```

17.4 Certificate Management

This is not a tutorial on X.509 certificates and their management. Refer to the listed references, Section 17.6, 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. It's 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, it's 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 SSL for OpenVMS Alpha/IA64/VAX product provides the ''SSL Certificate Tool'' procedure can be used to perform most required certificate management tasks from a menu-driven interface.

```
$ @SSL$COM:SSL$CERT_TOOL.COM

        S S L   C e r t i f i c a t e   T o o l

                        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. Hash Certificates
    8. Hash Certificate Revocations
    9. Exit

   Enter Option:
```

- The WASD OpenSSL kit provides elementary DCL procedures and brief notes in the HT_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.

17.4.1 Server Certificate

The server uses a certificate to establish it's 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 SSL package. If this is not available (for instance when using the HP SSL for OpenVMS Alpha/IA64/VAX 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. Both Netscape Navigator (v3.*n* & v4.*n*, v5.*n*, v6.*n*) and MS Internet Explorer (v4.*n*, v5.*n*) automatically invokes a server certificate load dialog when it encounters 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.

Navigator should be able to load using either certificate format. MSIE v3.*n* will load and report on the ''.CRT'' certificate quite contentedly, but then will not allow it to be used because it does not represent a well-known Certficate Authority. MSIE v4.*n* and v5.*n* seem able to use the ''.CRT'' certificate.

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.

17.4.2 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.

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 HT_ROOT:[SRC.OPENSSL-n_n_n].WASDCLIENT_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 it's 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 it's certificate database.

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

17.4.3 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 SSL for OpenVMS Alpha/IA64/VAX product ''SSL Certificate Tool'' described in Section 17.4 a CSR can easily be generated using it's 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 SSL for OpenVMS Alpha/IA64/VAX product).

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

```
$ SET DEFAULT HT_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 == "$HT_ROOT:[SRC.OPENSSL-version.AXP.EXE.APPS]OPENSSL.EXE"
```

   When using the HP SSL for OpenVMS Alpha/IA64/VAX product or other OpenSSL toolkit the verb may already be available.

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

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

```
$ RANDFILE = "HT_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.

```
HT_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 -
HT_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 HT_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 17.3.4). 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 == "$HT_ROOT:[SRC.OPENSSL-version.AXP.EXE.APPS]OPENSSL.EXE"
   ```

   When using the HP SSL for OpenVMS Alpha/IA64/VAX product or other OpenSSL toolkit the verb may already be available.

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

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

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

3. 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
   ```

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

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

5. Delete the temporary file.

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

17.5 SSL CGI Variables

CGI variables specific to SSL transactions optionally may be enabled using HTTPD$MAP mapping rules. See Section 13.4.5 in Chapter 13. 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=purveyor

```
WWW_SECURITY_STATUS == "SSL"
WWW_SSL_CIPHER == "RC4-MD5"
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 HTTPd CA Cert
/OU=OpenSSL 0.9.6 Testing Only/CN=WASD VMS Hypertext Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_DN == "/C=AU/ST=SA/L=Adelaide/O=WASD HTTPd Server Cert
/OU=OpenSSL 0.9.6 Testing Only/CN=WASD VMS Hypertext Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_VERSION == "SSLv3"
```

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

set /path/* SSLCGI=apache_mod_ssl

```
WWW_SSL_CIPHER == "RC4-MD5"
WWW_SSL_CIPHER_ALGKEYSIZE == "128"
WWW_SSL_CIPHER_USEKEYSIZE == "128"
WWW_SSL_PROTOCOL == "SSLv3"
WWW_SSL_SERVER_A_KEY == "rsaEncryption"
WWW_SSL_SERVER_A_SIG == "md5WithRSAEncryption"
WWW_SSL_SERVER_I_DN == "/C=AU/ST=SA/L=Adelaide/O=WASD HTTPd CA Cert
/OU=OpenSSL 0.9.6 Testing Only/CN=WASD VMS Hypertext Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_I_DN_C == "AU"
WWW_SSL_SERVER_I_DN_CN == "WASD VMS Hypertext 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 HTTPd CA Cert"
WWW_SSL_SERVER_I_DN_OU == "OpenSSL 0.9.6 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 HTTPd Server Cert
/OU=OpenSSL 0.9.6 Testing Only/CN=WASD VMS Hypertext Services
/Email=Mark.Daniel@wasd.vsm.com.au"
WWW_SSL_SERVER_S_DN_C == "AU"
WWW_SSL_SERVER_S_DN_CN == "WASD VMS Hypertext 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 HTTPd Server Cert"
WWW_SSL_SERVER_S_DN_OU == "OpenSSL 0.9.6 Testing Only"
WWW_SSL_SERVER_S_DN_ST == "SA"
WWW_SSL_SERVER_V_END == "Sep 25 00:03:30 2005 GMT"
WWW_SSL_SERVER_V_START == "Sep 26 00:03:30 2000 GMT"
WWW_SSL_SESSION_ID == "344d1b01aa0636cb809eacf270279005f56cd5ebe1154569df810e56003ac70f"
WWW_SSL_VERSION_INTERFACE == "HTTPd-WASD/7.2.0 OpenVMS/AXP SSL"
WWW_SSL_VERSION_LIBRARY == "OpenSSL 0.9.6 24 Sep 2000"
```

The Apache *mod_ssl* client certificate details described in Section 17.3.11 above are not shown in the above example but would be included if the request was X.509 authenticated.

17.6 SSL References

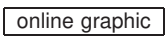
The following provide a starting-point for investigating SSL and OpenSSL further (verified available at time of publication).

- 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://www.openvms.compaq.com/openvms/products/ssl/ssl.html
  The HP SSL (Secure Sockets Layer) for OpenVMS Alpha/IA64/VAX product.
  The ‘‘Open Source Security for OpenVMS Alpha, Volume 2: Compaq SSL (Secure Sockets Layer) for OpenVMS Alpha’’ guide (see next link) available in PDF or HTML from this site (and the OpenVMS 7.3-1 and later documentation CD-ROM) is particularly comprehensive and relevant.

- http://www.openvms.compaq.com:8000/731final/6661/6661pro.htm
  http://www.openvms.compaq.com:8000/731final/documentation/pdf/ovms_731_open_source_ssl.pdf
  Open Source Security for OpenVMS Alpha, Volume 2: Compaq SSL (Secure Sockets Layer) for OpenVMS Alpha
- http://www.openvms.compaq.com/openvms/products/ips/apache/SSL_Resource_Guide.html
  The CSWS SSL Resource Guide. Contains links to a broad range of subjects and organisations.
- http://www.mozilla.org/projects/security/pki/nss/ref/ssl/
  Mozilla.org's SSL Reference
- http://www.rsasecurity.com/standards/ssl/basics.html
  RSA's SSL Basics for Internet Users
- http://www.rsasecurity.com/rsalabs/faq/
  ftp://ftp.rsasecurity.com/pub/labsfaq/rsalabs_faq41.pdf
  RSA's FAQ on Cryptography
- http://docs.sun.com/source/816-6156-10/
  Sun Microsystems' Introduction to SSL
- http://docs.sun.com/source/816-6154-10/
  Sun Microsystems' Introduction to Public-Key Cryptography
- http://www.cs.auckland.ac.nz/~pgut001/links.html
  ''Encryption and Security-related Resources'' . . . an (almost rediculously) exhaustive list of security and cryptography links, including some on SSL.

Chapter 18
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.



The value of the WATCH facility Chapter 19 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 22.

Access to many portions of the package is constrained by file protections and directory listing access files. See Section 15.10.8 for a method for circumventing these restrictions.

18.1 Access Without 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 19). This can be done quite simply by using the authentication skeleton-key (Section 15.11). This allows the site administrator to register a username and password from the commnd-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 == "$HT_EXE:HTTPD.EXE"
$! HTTPD == "$HT_EXE:HTTPD_SSL.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
```

18.2 Access Configuration

One established the site should make the Server Administration facility a configured facility of the site. The value of it's facilities cannot be overstated. The section Section 14.1 provides a short guide to setting up authorization for server administration purposes.

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 15.10.5.

```
[VMS]
/httpd/-/admin/*  #localhost,~username,r+w
```

If SSL is in use (Chapter 17) 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 HTTPD$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 22. 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 HT_ROOT:[LOCAL]).

```
[VMS]
/httpd/-/admin/*  #localhost,~username,r+w
/ht_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
```

18.3 Server Instances

With a single instance (Section 6.2) access to Server Administration reports, etc. is always serviced by the one server process. If multiple instances are configured in common with all requests administration requests will be serviced by any one of the associated processes depending on the indeterminate current 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 (Administration Services) available from the Server Administration page.
2. The Server Administration page (Control Section) and the command-line (Section 5.5.2.6) 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 HTTPD$CONFIG configuration directive [InstanceMax] to once again determine the number of instances required.

The latter approach is particularly useful when performing detailed WATCH activities (Chapter 19).

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 wasd.dsto.defence.gov.au: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 wasd.dsto.defence.gov.au:80
Server Administration - HTTPd:80
```

18.4 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. These are listed 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 18.1).

- **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$OUTPUT) log.
- **Configuration -** A tabular summary of the server's current configuration. This is a convenient method for viewing the information from the HTTPD$CONFIG 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 15) 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.
- **Cache -** Allows monitoring of cache behaviour and performance, as well as the files currently in the cache (Chapter 20).
- **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).

  Subprocess information shows how many actual subprocesses 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 subprocesses may actually exist at any one time (the margin allows for subprocess deletion latency). A count of how many times the CGIplus subprocesses 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 subprocess life-time (in minutes, zero implying indefinite), the number purged due to life-time expiry and the number of CGIplus subprocesses that the server has actually purged (deleted) to maintain the soft-limit margin specified above.

Each of the allocated subprocess 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 subprocess existing. This can be a zombie subprocess or a CGIplus subprocess. If no subprocess is indicated then the other information represents the state the last time the item's associated subprocess 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 subprocess 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 subprocess is currently in use the client information show the client host name.

If any subprocesses are associated with any data structure a *purge* button is provided that forces all subprocesses 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 subprocess deletion occurs when that processing is complete. The purge button **does not force** a subprocess to delete, so a second button **forces** all subprocesses 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.

- **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 (Chapter 11) 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 (Section 11.4) 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 current requests (always shows at least your own connection accessing this report :^) and if enabled by configuration an optional history list 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 19).

  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.

- **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).

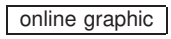
  startup - green
  shutdown - black
  restart - grey
  error exit - red

  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.



- **WATCH -** This report provides an online, real-time, in-browser-window view of request processing on the **running server**. See Chapter 19 for details.

18.5 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 it's 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 18.6).
- **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 18.6).
- **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 18.6).
- **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 15 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 HTTPD$SITELOG (Logical Names).

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 (Section 7.5) this poses minimal threat to server configuration integrity.

1. Specifically map the /ht_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 /ht_root/ ones).

```
# HTTPD$MAP
pass /ht_root/local/* auth=all
```

2. Add appropriate authorization rules (example from Section 14.1).

```
# HTTPD$AUTH
["Web Admin"=WASD_WEBADMIN=id]
/httpd/-/admin/* r+w
/ht_root/local/* r+w
```

3. Update access to the directory can be applied using the SECHAN utility (Section 23.11).

```
$ SECHAN /WRITE HT_ROOT:[000000]LOCAL.DIR
$ SECHAN /WRITE HT_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 15.10.7) 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.

```
# HTTPD$MAP
set /ht_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 HT_ROOT:[000000]LOCAL.DIR
$ SECHAN /PACKAGE HT_ROOT:[LOCAL]
$ SECHAN /CONTROL HT_ROOT:[000000]LOCAL.DIR
```

18.6 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 5.5.2.

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 HTTPD$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.

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.

Chapter 19
WATCH Facility

The WATCH facility is a powerful adjunct in server administration. From the Server Administration facility (Chapter 18) 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.

19.1 Server Instances

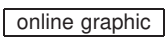
With a single instance (Section 6.2) access to WATCH is always through the one server process. If multiple instances are configured WATCH requests, in common with all others, will be serviced by any one of the associated processes depending on the indeterminate current state of the round-robin distribution.

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.

There is no simple solution for this issue as the round-robin distribution of requests is determined by the network device driver. When debugging server or script behaviour often the simplest solution is to temporarily reset to one the number of instances running on the system. See Section 18.3, Section 18.6 and Section 5.5.2.6.

19.2 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 subprocess (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.

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.

Other

- **Match -** Shows a significant level of detail during string matching activities. May be useful during mapping, authorization and conditional processing.
- **Logging -** Access logging events include log open, close and flush, as well as request entries.
- **SSL -** If the Secure Sockets Layer image is in use this category provides a indication of high-level activity.
- **Quotas -** Display available server process resource quotas with significant events.

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.

19.3 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.

- **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.
- **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 (Section 6.11.6) enter the track’s unique identifier string preceded by a dollar symbol (e.g. ‘‘$ORoKJAOef8sAAAkuACc’’).

These filters are controlled using fully-specified or wildcarded strings. Requests that do not match the filter are not reported. In the case of originating client and destination service, requests are eliminated before ever appearing in the report. Path and track filtering is slightly different, requiring some request processing before the target can be determined. Depending on the categories selected this may result in some events begin displayed. It will be eliminated, with an accompanying explanatory message, as soon the path or track identifier has been determined.

The following examples are grouped in the same order as the categories listed above; client, service and path.

```
alpha.wasd.dsto.defence.gov.au
*.wasd.dsto.gov.au
131.185.250.202
131.185.250.*

beta.wasd.dsto.defence.gov.au:8000
beta.wasd.dsto.defence.gov.au:*
http://*
https:*
*:80

/ht_root/src/*
/cgi-bin/*
/web/*/cyrillic/*
$ORoKJAOef8sAAAkuACc
http://proxied.host.name/*
```

19.4 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 it's 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.

```
29-MAY-2003 08:54:59  WATCH REPORT  wasd.dsto.defence.gov.au:80
------------------------------------------------------------
HTTPd-WASD/8.3.0 OpenVMS/AXP SSL (26-MAY-2003 08:07:52.96)
Compaq TCPIP$IPC_SHR V5.3-18 (29-MAR-2002 05:01:04.60)
OpenSSL 0.9.7b 10 Apr 2003 (12-APR-2003 09:31:08.21)
$ CC (V7.3-1/60590001) /DECC /STAND=RELAXED_ANSI /PREFIX=ALL /OPTIMIZE /NODEBUG /WARNING=(N
AlphaServer 1200 5/533 4MB with 1 CPU and 256MB running VMS V7.3-1 (ODS-5 enabled, VMS NAML
$ HTTPD /PRIORITY=4 /SYSUAF=ID/PROFILE/PERSONA
AST:1993/2000 BIO:1996/2000 BYT:671728/749424 DIO:1000/1000 ENQ:451/500 FIL:291/300 PGFL:47
DCL Scripting: detached, as HTTP$NOBODY, PERSONA enabled
Process: HTTPd:80 HT_ROOT:[STARTUP]STARTUP_SERVER.COM;5 HT_ROOT:[LOG_SERVER]WASD_2003052009
Instances: WASD::HTTPd:80
Watching: connect, request, req-header, response, res-header, error (603)
Client: "" Service: "" Path: ""
|Time_______|Module__|Line|Item|Category__|Event...|
|08:55:06.74 NET      1839 0001 CONNECT    ACCEPTED wasd,49846 on http://131.185.30.1:80 BG
|08:55:06.75 REQUEST  1751 0001 REQ-HEADER HEADER 608 bytes|
GET /web/colloquia/ HTTP/1.1
Host: wasd.dsto.defence.gov.au
User-Agent: Mozilla/5.0 (X11; U; OpenVMS AlphaServer_1200_5/533_4MB; en-US; rv:1.3) Gecko/2
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,vid
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate,compress;q=0.9
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://wasd.dsto.defence.gov.au/home.html
If-Modified-Since: Fri, 10 Nov 2000 06:33:58 GMT
Cache-Control: max-age=0
```

```
|08:55:06.75 NET      1313 0001 CONNECT    VIRTUAL wasd.dsto.defence.gov.au:80|
|08:55:06.75 REQUEST  2701 0001 REQUEST    GET /web/colloquia/|
|08:55:06.75 CACHE    0466 0001 RESPONSE   CACHE search 8EE655F806C11A98E0B51D52FFE5B269|
|08:55:06.77 CACHE    0712 0001 RESPONSE   CACHE revalidate WEB:[COLLOQUIA]INDEX.HTML;|
|08:55:06.77 CACHE    0717 0001 RESPONSE   CACHE hit volatile WEB:[COLLOQUIA]INDEX.HTML;|
|08:55:06.77 NET      2167 0001 RES-HEADER HEADER 249 bytes|
HTTP/1.0 200 OK
Server: HTTPd-WASD/8.3.0 OpenVMS/AXP SSL
Date: Wed, 28 May 2003 23:25:06 GMT
Last-Modified: Fri, 10 Nov 2000 06:33:58 GMT
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 1596
Connection: Keep-Alive
Keep-Alive:
|08:55:06.77 REQUEST  0682 0001 REQUEST    STATUS 200 rx:608 tx:1845 bytes 0.0208 seconds|
|08:55:06.77 REQUEST  0326 0001 CONNECT    KEEP-ALIVE wasd,49846|
|08:55:12.84 NET      2027 0001 CONNECT    CLOSE wasd,49846|
|08:55:29.46 end|
```

19.5 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. 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.

19.6 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 20
Cache

WASD HTTPd provides an optional, configurable, monitorable file data and revision time cache. File data, so that requests for documents can be fulfilled without reference to the underlying file system, potentially reducing request latency and more importantly improving overall server performance and system impact, and file revision time, so that requests specifying an ''If-Modified-Since:'' header can also benefit from the above. Files are cached using a hash derived from the VMS file-system path equivalent generated during the mapping process (i.e. represents the file name) but before any actual RMS activity. WASD can also cache the content of responses from non-file sources. This can be useful for reducing the system impact of frequently accessed, dynamically generated, but otherwise relatively static pages. These sources are cached using a hash derived from virtual service connected to and the request URI.

Why Implement Caching?

Caching, in concept, attempts to improve performance by keeping data in storage that is faster to access than it's usual location. The performance improvement can be assessed in three basic ways; reduction of

- response when accessing the data (latency and transfer time)
- processing involved (CPU cycles)
- impact on the usual storage location (file system I/O)

This cache is provided to address all three. Where networks are particularly responsive a reduction in request latency can often be noticeable. It is also suggested a cache ''hit'' may consume less CPU cycles than the equivalent access to the (notoriously expensive) VMS file system. Where servers are particularly busy or where disk subsystems particularly loaded a reduction in the need to access the file system can significantly improve performance while simultaneously reducing the impact of the server on other system activities.

A comparison between cached and non-cached performance is provided in Chapter 21.

Terminology

Term	Description
hit	Refers to a request path being found in cache. If the data is still valid the request can be supplied from cache.
flushing	Occurs when the cache becomes full, with older, less frequently used cache entries being removed from the cache and replaced by other files.
loading	Refers to reading the contents of a file into cache memory.
permanent	These entries are loaded once and remain in the cache until it is explicitly purged by the administrator or the the server is restarted. They are not flushed or revalidated.
revalidate	Compare the cache entrys size and modification date-time to the file it represents in the file-system. Obviously a difference indicates the content has changed.
valid	The file from which the cached data was originally read has not had it's revision date changed (the implication being the file contents have not changed).
volatile	Entries have the original file periodically checked for modification and are reloaded if necessary. They can also be flushed if demand for space requires it.

20.1 Non-File Content Caching

The WASD cache was originally provided to reduce file-system access (a somewhat expensive activity under VMS). With the expansion in the use of dynamically generated page content (e.g. PHP, Perl, Python) there is an obvious need to reduce the system impact of some of these activities. While many such responses have content specific to the individual request a large number are also generated as general site pages, perhaps with simple time or date components, or other periodic information. Non-file caching is intended for this type of dynamic content.

Revalidation of non-file content is fraught with a number of issues and so is not provided. Instead the cache entry is flushed on expiry of the [CacheValidateSeconds], or as otherwise specified by path mapping, and the request is serviced by the content source (script, PHP, Perl, etc.) with the generated response being freshly cached. All of the considerations described in Section 20.4 apply equally to file and non-file content.

Controlling Non-File Content Caching

Determining which non-file content is cached and which not, and how long before flushing, is done using mapping rules (Section 13.4.5). The source of non-file cache content is specified using one or a combination of the following SET rules against general or specific paths.

cache=[no]cgi from Common Gateway Interface (CGI) script response
cache=[no]file from the file system (default and pre-8.4 cache behaviour)
cache=[no]net caches the full data stream irrespective of the source
cache=[no]nph full stream from Non-Parse Header (NPH) script response
cache=[no]query cache requests with query strings (**use with care**)

cache=[no]script both CGI and NPH script responses
cache=[no]ssi from Server-Side Includes (SSI) documents

A good understanding of site requirements and dynamic content sources, along with considerable care in specifying cache path SETings, is required to cache dynamic content effectively. It is especially important to get the content revalidation period appropriate to the content of the pages. This is specified using the following path SETings.

cache=expires=0 cancels any expiry
cache=expires=DAY expires when the day changes
cache=expires=HOUR when the clock hour changes
cache=expires=MINUTE when the clock minute changes
cache=expires=<hh:mm:ss> expires after the specified period in the cache

For example. To cache the content of PHP-generated home pages that contain a time-of-day clock, resolving down to the minute, would require a mapping rule similar to the following.

```
set /**/index.php cache=cgi cache=expires=minute
```

20.2 Permanent and Volatile

The WASD file cache provides for some resources to be permanently cached while others are allowed to be moved into and out of the cache according to demand. Most sites have at least some files that are fundamental components of the site's pages, are rarely modified, commonly accessed, and therefore should be permanently available from cache. Other files are modified on a regular or ad hoc basis and may experience fluctuations in demand. These more volatile resources should be cached based on current demand.

Volatile caching is the default with the site administrator using mapping rules to indicate to the server which resources on which paths should be permanently cached (Section 20.5).

Although permanent and volatile entries share the same cache structure and are therefore subject to the configuration's maximum number of cache entries, the memory used store the cached file data is derived from separate pools. The total size of all volatile entries data is constrained by configuration. In contrast there is no configuration limit placed on the quantity of data that can be cached by permanent entries. One of the purposes of the permanent aspect of the cache is to allow the site administrator considerable discretion in the configuration of the site's low-latency resources, no matter how large or small that might be. Of course there is the ultimate constraint of server process and system virtual memory limits on this activity. It should also be kept in mind that unless sufficient physical memory is available to keep such cached content in-memory the site may only end up trading file-system I/O for page file I/O.

20.3 Cache Suitability Considerations

A cache is not always of benefit! It's cost may outweigh it's return.

Any cache's efficiencies can only occur where subsets of data are consistently being demanded. Although these subsets may change slowly over time a consistent and rapidly changing aggregate of requests lose the benefit of more readily accessable data to the overhead of cache management, due to the constant and continuous flushing and reloading of cache data. This server's cache is no different, it will only improve performance if the site experiences some consistency in the files requested. For sites that have only a small percentage of files

being repeatedly requested it is probably better that the cache be disabled. The other major consideration is available system memory. On a system where memory demand is high there is little value in having cache memory sitting in page space, trading disk I/O and latency for paging I/O and latency. On memory-challenged systems cache is probably best disabled.

To help assessment of the cache's efficiency for any given site monitor the Server Administration facility's cache report.

Two sets of data provide complementary information, cache activity and file request profile.

- **Activity Data**

  This summarizes the cache search behaviour, in particular that of the hash table.

  The ''searched'' item, indicates the number of times the cache has been searched. Most importantly, this may include paths that can never be cached because they represent non-file requests (e.g. directory listings). Requests involving scripts, and some others, never attempt a cache search.

  The ''hit'' item, indicates the number of times the hash table directly provided a cached path. This is very efficient.

  The ''miss'' item, indicates the number of times the hash table directly indicated a path was not cached. This is decisive and is also very efficient.

  The ''collision'' item, indicates the number of times multiple paths resolved to the same hash table entry. Collisions require further processing and are far less efficient. The sub-items, ''collision hits'' and ''collision misses'' indicate the number of times that further processing resulted in a found or not-found cache item.

  A large number of cache misses compared to searches may only indicate a large number of non-cacheable requests and so depending on that further datum is not of great concern. A large proportion of collisions (say greater than 12.5%) is however, indicating either the hash table size needs increasing (1024 should be considered a minimum) or the hashing algorithm in the software need reviewing :^)

- **Files Data**

  This summarizes the site's file request profile.

  With the ''loads not hit'' item, the count represents the cumulative number of files loaded but never subsequently hit. If this percentage is high it means most files loaded are never hit, indicating the site's request profile is possibly unsuitable for caching.

  The item ''hits'' respresents the cumulative, total number of hits against the cumulative, total number of loads. The percentage here can range from zero to many thousands of percent :^) with less than 100% indicating poor cache performance and from 200% upwards better and good performance. The items ''1-9'', ''10-99'' and ''100+'' show the count and percentage of total hits that occured when a given entry had experienced hits within that range (e.g. if an entry has had 8 previous hits, the ninth increments the ''1-9'' item whereas the tenth and eleventh increments the ''10-99'' item, etc.)

Other considerations also apply when assessing the benefit of having a cache. For example, a high number and percentage of hits can be generated while the percentage of ''loads not hit'' could be in the also be very high. The explanation for this would be one or two frequently requested files being hit while most others are loaded, never hit, and flushed as other files request cache space. In situations such as this it is difficult to judge whether cache processing is improving performance or just adding overhead.

20.4 Cache Content Validation

The cache will automatically revalidate the volatile entry file data after a specified number of seconds ([CacheValidateSeconds] configuration parameter), by comparing the original file revision time to the current revision time. If different the file contents have changed and the cache contents declared invalid. If found invalid the file transfer then continues outside of the cache with the new contents being concurrently reloaded into the cache. Permanent entries are not subject to revalidation and it's associated reloading.

Cache validation is also always performed if the request uses ''Cache-Control:'' with *no-cache*, *no-store* or *max-age=0* attributes (HTTP/1.1 directive), or if a ''Pragma: no-cache'' field (HTTP/1.0 directive). These request directives are often associated with a browser agent *reload page* function. Hence there is no need for any explicit flushing of the cache under normal operation. If a document does not immediately reflect any changes made to it (i.e. validation time has not been reached) validation (and consequent reload) can be ''forced'' with a browser reload. Permanent entries are also not subject to this source of revalidation. The configuration directive [CacheGuardPeriod] limits this form of revalidation when used within the specified period since last revalidated. It has a default value of fifteen seconds.

If a site's contents are relatively static the validation seconds could be set to an extended period (say 3600 seconds, one hour) and then rely on an explicit ''reload'' to force validation of a changed file.

The entire cache may be purged of cached data, both volatile and permanent entries, either from the Server Administration facility or using command line server control.

```
$ HTTPD /DO=CACHE=PURGE
```

20.5 Cache Configuration

The cache is controlled using HTTPD$CONFIG configuration file and HTTPD$MAP mapping file directives. A number of parameters control the basics of cache behaviour.

- **[Cache]** enables and disables caching.
- **[CacheEntriesMax]** and **[CacheTotalKBytesMax]** provide growth limits to cache expansion. Maximum entries limits the number of files loaded into the cache before entries begin to be reused (flushing the original contents). Maximum total kilobytes allocated to the cache provides a ceiling on the memory consumed. These parameters operate to limit each other (i.e. if one reaches it's limit before the other, the other will not grow further either).
- **[CacheFileKBytesMax]** provides a limit on file size (in kilobytes). Files larger than the specified limit will not be cached. This may be overridden on a per-path basis using the *set cache=max=<integer>* mapping rule (see below).

- **[CacheFrequentIntervals]** and **[CacheFrequentSeconds]** attempt to reduce unproductive reuse of cache entries by providing the cache with some indication of what constitutes a frequently hit entry. If it is frequently hit then it should not be immediately reused when there is a demand for cache space. The first parameter sets the number of hits an entry must sustain before being a candidate for *CacheFrequentSeconds* assessment. If a file has been hit at least *CacheFrequentIntervals* times in total and the last hit was within the number of seconds set by *CacheFrequentSeconds* it will not be flushed and reused. If it has not been hit within the specified period it will be reused.
- **[CacheGuardPeriod]** prevents browser initiated content revalidation described above (Section 20.4). It is provided to help limit unnecessary file-system activity. It's default is fifteen seconds.
- **[CacheHashTableEntries]** *(obsolete)*
- **[CacheValidateSeconds]** The interval after which a cache entry's original, content revision time is revalidated against the file's current revision time. If not the same the contents are declared invalid and reloaded. Setting this to a greater period reduces disk I/O but revised files may not be obvious within an acceptable timer unless a revalidation is forced with a *reload*. Permanent entries are not subject to validation.

Mapping Rules

Mapping rules (Section 13.4.5) allow further tailoring of cache behaviour based on request (file) path. Those files that should be made permanent entries are indicated using the *cache=perm* directive. In the following example all files in the WASD runtime directories (directory icons, help files, etc.) are made permanent cache entries at the same time the path is mapped.

```
pass /*/-/* /ht_root/runtime/*/* cache=perm
```

Of course, specified file types as well as specific paths can be mapped in this way. Here all files in the site's /help/ path are made permanent entries except those having a .PS type (PostScript documents).

```
set /help/* cache=perm
set /help/*.ps cache=noperm
```

The configuration directive [CacheFileKBytesMax] puts a limit on individual file size. Those exceeding that limit are considered too large and not cached. It is possible to override this general constraint by specifying a maximum size (in kilobytes) on a per-path basis.

```
set /help/examples*.jpg cache=max=128
set /cai/*.mpg cache=max=2048 cache=perm
```

Caching may be disabled and/or enabled for specified paths and subpaths.

```
set /web/* cache=none
set /web/icons/* cache
```

20.6 Cache Control

The cache may be enabled, disabled and purged from the Server Administration facility (Chapter 18). In addition the same control may be exercised from the command line (Section 5.5.2) using

```
$ HTTPD /DO=CACHE=ON
$ HTTPD /DO=CACHE=OFF
$ HTTPD /DO=CACHE=PURGE
```

If cache parameters are altered in the configuration file the server must be restarted to put these into effect. Disabling the cache on an ad hoc basis (from menu or command line) does not alter the contents in any way so it can merely be reenabled with use of the cache's previous contents resuming. In this way comparisions between the two environments may more easily be made.

20.7 Circumventing The Cache

There are often good reasons for bypassing or avoiding the cache. For instance, where a document is being refreshed within the cache revalidation period specified by [CacheValidateSeconds] (Section 20.4). There are two mechanisms available for bypassing or invalidating the file cache.

1. This directs the server to always get the file from the file-system.

   ```
   SET /path/not/to/cache/* cache=none
   ```

2. Specify a version component when requesting the file. WASD never caches a file if the request contains a version component. It does not need to be a full version number, a semi-colon is sufficient. For example:

   ```
   /ht_root/robots.txt;
   ```

Chapter 21
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.

The server has been tested with up to 60 concurrent requests originating from 6 different systems and continues to provide an even distribution of data flow to each client (albeit more slowly :^) The author has also observed sites where traffic levels frequently reach 50 to 100 concurrent requests both for standard and proxy serving.

The test system was a lightly-loaded AlphaServer 4100 4/400 (4 x 400MHz CPUs), VMS v7.3-1 and DEC TCP/IP 5.3. No *Keep-Alive:* functionality was employed so each request required a complete TCP/IP connection and disposal. DNS (name resolution) and access logging were disabled. The server and test-bench utility were located on separate systems with 100 Mbps Fast-Ethernet interconnection.

As of v7.1 the performance data is collected using the ''ApacheBench'' utility (Section 23.5). DCL procedures with sets of ApacheBench calls are used to benchmark requests. These procedures and the generated output from benchmark runs (collected via $@*procedure*/OUTPUT=*filename*) are available in the HT_ROOT:[EXERCISE] directory.

These results are indicative only!

On a clustered, multi-user system too many things vary slightly all the time. Hence the batching of accesses, interleaved between servers, attempting to provide a representative result.

OSU/CSWS Comparison

Until v5.3 a direct comparison of performance between OSU and WASD had not been made (even to satisfy the author's own occasional curiosity). After a number of users with experience in both environments commented . . . WASD *seemed* faster, was it? . . . it was decided to make and provide comparisons using the same metrics used on WASD for some time.

Every endeavour has been made to ensure the comparison is as equitable as possible (e.g. each server executes at the same process priority, has a suitable cache enabled, 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. Tests showing a port 7080 were to WASD, port 7777 to the OSU server, and port 8888 to Apache. All servers were configured ''out-of-the-box'', minimal changes (generally just path mappings), WASD executing via the [INSTALL]DEMO.COM procedure.

Of course performance is just one of a number of considerations in any software environment (otherwise we wouldn't be using VMS now would we? ;^) No specific conclusions are promoted by the author. Readers may draw their own from the results recorded below.

For this document the results were derived using the WASD v8.3, CSWS V1.3 (VMS Apache 1.3.26), and OSU 3.10 servers.

21.1 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.

Cache Disabled - Requests/Second

Response	Concurrent	WASD	OSU	CSWS
0K	1	181	117	34
0K	10	242	130	35
64K	1	70	40	36
64K	10	92	53	34

Cache Enabled - Requests/Second

Response	Concurrent	WASD	OSU	CSWS
0K	1	549	444	33
0K	10	879	533	34
64K	1	104	72	36
64K	10	123	85	25

Result file:

HT_ROOT:[EXERCISE]PERF_FILES_NOCACHE_AB_V83.TXT
HT_ROOT:[EXERCISE]PERF_FILES_AB_V83.TXT

With both WASD cached and non-cached **throughput actually improves** at ten concurrent requests (undoubtably due to the latency of the serial TCP/IP connection/disconnection in one-by-one, compared to several happening concurrently).

Note that the response and transfer benefits decline noticably with file size (transfer time). The difference between cached and non-cached with the zero file size (no actual data transfer involved) gives some indication of the raw difference in response latency, some 300-400% improvement. This is a fairly crude analysis, but does give some indication of cache efficiencies.

Just one other indicative metric of the two servers, CPU time consumed during the file measurement 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	CSWS
Disabled	21	61	-
Enabled	12	25	-

File Transfer Rate

Under similar conditions results indicate a potential transfer rate **well in excess of 1 Mbyte per second**. This serves to demonstrate that server architecture should not be the limiting factor in file throughput.

Transfer Rate - MBytes/Second

Response	Concurrent	WASD	OSU	CSWS
3.9MB (7700 blocks)	1	2.09	1.37	2.14
3.9MB (7700 blocks)	10	1.68	1.36	1.84

Result file:

HT_ROOT:[EXERCISE]PERF_XFER_AB_V83.TXT

Significantly, there were no dramatic drops in transfer rate between one and ten concurrent requests! The results for Apache indicate one occasion where a collection of child processes performs very well (with assistance from generous VCC_ . . . cache settings).

File Record Format

The 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!

So significant is this efficiency improvement a module exists to automatically convert VARIABLE record files to STREAM-LF when detected by the file transfer module. This is disabled by default but the user is strongly encouraged to enable it and to ensure that stream format files are provided to the server by other hypertext generating and processing utilitites.

21.2 Scripting

Persistant-subprocesses are probably the most efficient solution for child-process scripting under VMS. See ‘‘Scripting Environment’’ document. The I/O still needs to be on-served to the client by the server.

A simple performance evaluation shows the relative merits of the four WASD scripting environments available, plus a comparison with OSU and Apache. HT_ROOT:[SRC.CGIPLUS]CGIPLUSTEST.C, which executes in both standard CGI and CGIplus environments, and an ISAPI example DLL, HT_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.

DECnet-based scripting was tested using essentially the same environment as subprocess-based CGI, assessing the performance of the same script being executed using DECnet to manage the processes. Three separate environments have been evaluated, WASD-DECnet-CGI, WASD-OSU-emulation and OSU. The OSU script used the WASD CGISYM.C utility to generate the required CGI symbols (also see WASD/OSU Comparison). DECnet-Plus T5.0.3 was in use.

CGI Scripting - Requests/Second

Response	Concurrent	CGI	CGIplus	ISAPI	DECnet-CGI	OSU-emul	OSU	CSWS
0KB	1	32	400	387	14	12	10	4
0KB	10	78	549	549	18	20	17	7
64KB	1	26	97	98	11	10	7	4

Response	Concurrent CGI		CGIplus	ISAPI	DECnet-CGI	OSU-emul	OSU	CSWS
64KB	10	30	49	48	13	15	11	7

Result file:

HT_ROOT:[EXERCISE]PERF_SCRIPTS_AB_V83.TXT

Scripting Observations

Although these results are indicative only, they do show CGIplus and ISAPI to have a potential for improvement over standard CGI from a factor of 5 (500%) up to factors in excess of 10 (1000%) - 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/application has a large activation time the reduction in response latency could be even more significant** (e.g. Perl scripts and RDMS access languages).

CGIplus under V7.2 has seen a dramatic increase in throughput over previous version benchmarks . . . in excess of a factor of 2 (100%)! This is entirely due to the new ''struct'' mode available. See the Scripting Overview for further detail.

DECnet Observations

This section comments on non-persistant scripts (i.e. those that must run-up and run-down with each request - general CGI behaviour). Although not shown here measurements of connection reuse show significant benefits in reduced response times, consistency of response times and overall throughput, showing a difference of some 200% over non-reuse (similar improvements were reported with the OSU 3.3a server).

With ten simultaneous and back-to-back scripts and no connection reuse many more network processes are generated than just ten. This is due to the NETSERVER maintenance tasks such as log creation and purging, activating and deactivating the task, etc., adding latency into this script environment. The throughput was generally still lower than with subprocess-based scripting.

While earlier versions cautioned on the use of DECnet-based scripting this has been relaxed somewhat through connection reuse.

WASD/OSU Comparison

A direct comparison of CGI performance between WASD and OSU scripting is biased in favour of WASD, as OSU scripting is based on it's own protocol with CGI behaviour layered-in above scripts that require it. Therefore a non-CGI comparison was devised. The script is designed to favour neither environment, merely return the plain-text string ''Hello!'' as quickly as possible. Data for Apache is also included, although this type of scripting is not really it's forte.

```
$! OSU and WASD scripting face-to-face in a script that favours neither unduly
$ if f$type(WWWEXEC_RUNDOWN_STRING) .nes. ""
$ then
$    write net_link "<DNETTEXT>"
$    write net_link "200 Success"
$    write net_link "Hello!"
$    write net_link "</DNETTEXT>"
$ else
$    write sys$output "Content-Type: text/plain"
$    write sys$output ""
$    write sys$output "Hello!"
$ endif
```

Face-to-Face - Requests/Second

	Concurrent	CGI	CGIplus	ISAPI	DECnet-CGI	OSU-emul	OSU	CSWS
''Hello!''	1	61	n/a	n/a	n/a	n/a	23	5
''Hello!''	10	137	n/a	n/a	n/a	n/a	33	8

Result file:

HT_ROOT:[EXERCISE]PERF_SCRIPTS_AB_V83.TXT

WASD/CSWS Scripting Comparison

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 CSWS (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 CSWS, Perl and PHP. The engines used in both server environments were identical. WASD 8.3 with PHPWASD112 and PERLRTE121 packages. CSWS 1.3 with CSWS_PHP-V0101 and PERL-V0506-1-1 packages.

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 it's 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. No data is provided for the OSU package.

Persistent Scripting - Requests/Second

	Concurrent	WASD	CSWS
face2face.pl	1	92	17
face2face.pl	10	241	41
face2face.php	1	73	21
face2face.php	10	342	39
php_info.php	1	57	18
php_info.php	10	93	31

Result file:

HT_ROOT:[EXERCISE]PERF_PERSIST_AB_V83.TXT

Persistent Scripting Observations

These results demonstrate the efficiency and scalability of the WASD CGIplus/RTE technology used to implement it's 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 ''Scripting Environment'' document.

21.3 SSL

At this time there are no definitive measurements of SSL performance (Chapter 17). 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.

21.4 Suggestions

Here are some suggestions for improving the performance of the server, listed in approximate order of significance. 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. 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).
3. Use persistant-subprocess DCL/scripting (configuration parameter [ZombieLifeTime])
4. Ensure script processes are given every possible chance to persist (configuration parameter [DclBitBucketTimeout]).
5. Use the persistent scripting capabilities of CGIplus or ISAPI whenever possible.

6. Enable caching (configuration parameter [Cache]).
7. Ensure the server account's WSQUO and WSEXTENT quotas are adequate. A constantly paging server is a slow server!
8. 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.
9. Disable logging (configuration parameter [Logging]).
10. 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.
11. Reduce to as few as possible the number of mapping and authorization rules, particularly those that have conditions that require additional evaluation. Also see Chapter 13.
12. Use a pre-defined log format (e.g. ''common'', configuration parameter [LogFormat]). User-specified formats require more processing for each enrty.
13. Disable request history (configuration parameter [RequestHistory]).
14. Disable activity statistics (configuration parameter [ActivityDays]).

Chapter 22
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 (Chapter 17) 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 HTTPD$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 15.12 for information on write access control for the server account.

Chapter 23
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 HT_ROOT:[EXAMPLE]WASDVERBS.COM procedure.

```
$ AB == "$HT_EXE:AB"
$ HTTPD == "$HT_EXE:HTTPD"
$ HTTPDMON == "$HT_EXE:HTTPDMON"
$ MD5DIGEST == "$HT_EXE:MD5DIGEST"
$ QDLOGSTATS == "$HT_EXE:QDLOGSTATS"
$ SECHAN == "$HT_EXE:SECHAN"
$ STREAMLF == "@HT_EXE:STREAMLF"
$ WB == "$HT_EXE:WB"
```

23.1 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://wasd.dsto.defence.gov.au/echo/ht_root/
```

23.2 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 facility must be enabled through a mapping rule entry.

```
script /hiss/* /hiss/*
```

Usage details are described in Section 7.8.

23.3 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://wasd.dsto.defence.gov.au/where/ht_root/
```

23.4 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://wasd.dsto.defence.gov.au/xray/ht_root/
```

23.5 Apache Bench

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.
```

Apache Bench will only compile and run for Alpha, IA64 or VAX systems with VMS 7.*n* or greater available. Also see the WASD analogue, Section 23.14. Apache Bench 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.)

A small addition to functionality has been made. The WASD Apache Bench 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 it's use.

```
$ AB -H
$ AB -C 10 -N 100 http://the.server.name/ht_root/exercise/0k.txt
$ AB -C 50 -N 500 -K http://the.server.name/ht_root/exercise/64k.txt
$ AB -C 10 -N 100 http://the.server.name/cgi-bin/cgi_symbols
```

23.6 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 it's 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>]

/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 == "$HT_EXE:CALOGS"
$ CALOGS HT_LOGS:*200205*.LOG 2002_MAY.LOG
$ CALOGS /VERBOSE HT_LOGS:
$ CALOGS /NOWASD HT_LOGS:*200206*.LOG_* /OUTPUT=2002_JUNE.LOG
$ CALOGS /PROXY /NOWASD HT_LOGS:*2002*.LOG 2002_PROXY.LOG
```

23.7 HTAdmin

The HTAdmin utility assists in with the command-line maintenance of $HTA authorization databases (see Section 14.3 and Section 15.5).

$ htadmin <database> [<username>] [<qualifiers>]

/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
/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
```

23.8 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 elipsis).
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 (after WWWRKOUT server testing):

```
  WASD:: 1         HTTPDMON v2.0.0 AXP        Saturday, 12-MAY-2001 11:26:18

Process: HTTPd:80  PID: 2020022F  User: HTTP$SERVER  Version: 7.2
     Up: 0 21:05:51.88  CPU: 0 07:34:19.99  Startup: 1
Pg.Flts: 3526  Pg.Used: 37%  WsSize: 6402  WsPeak: 4729
    AST:  196/200   BIO: 1003/1020  BYT: 202270/202720  DIO: 498/500
    ENQ: 1989/2000  FIL:  298/300   PRC:    100/100      TQ:  26/30

Request: 184865  Current:16  Throttle: 0/0/0%  Peak: 29
 Accept: 178212  Reject:0  Busy:0  SSL: 17
CONNECT: 0  GET: 164717  HEAD: 20148  POST: 0  PUT: 0
  Admin: 0  Cache: 48/39992/0  DECnet: 3327/6655  Dir: 6670
    DCL: CLI:16730 CGI:9987 CGIplus:13310/13182 RTE:16/15 Processes: 168/5
   File: 3384/0  IsMap: 0  Proxy: 64911  Put: 0  SSI: 16657  Upd: 9988

    1xx: 0  2xx: 157816  3xx: 3  4xx: 18496 (403:27)  5xx: 1864  (none:0)
     Rx: 19,762,345  Tx: 1,053,847,437  (bytes)

  Proxy: enabled
CONNECT: 0  GET: 58720  HEAD: 6191  POST: 0  PUT: 0
    Not: Cacheable Request:14555 Response:23786
Network: Rx:118,463,055 Tx:7,208,716 (29%)  Accessed: 38356
 Lookup: Numeric:19 DNS:21 Cache:38317 Error:1
  Cache: Rx:2,863,617 Tx:320,164,012 (71%)  Read:26554/3 Write:10
 Device: DKA0: 4110480 blocks (2007MB)  Space: available
         2171744 used (1060MB 53%), 1938736 free (946MB 47%)
  Purge: 18 00:00:54, 98 files (2254/2412), 0 deleted (0/0)

   Time: 12 11:25:11  Status: 200  Rx: 95  Tx: 34479  Duration: 0.2400
Service: beta.dsto.defence.gov.au:80
   Host: beta.dsto.defence.gov.au (131.185.250.201)
Request: GET /ht_root/exercise/64k.txt
```

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 18).

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.

23.9 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 == "$HT_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.

23.10 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.

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 (Chapter 11).

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]

/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 HT_LOGS:*1999*.LOG /DATE="*/SEP/1999*"
```

- Records where the browser was an X-based Netscape Navigator

```
$ QDLOGSTATS HT_LOGS:*.LOG /USERAGENT=*MOZILLA*X11*
```

- Records of POST method requests

```
$ QDLOGSTATS HT_LOGS:*.LOG /METHOD=POST
```

- Records requesting a particular path

```
$ QDLOGSTATS HT_LOGS:*.LOG /PATH="/cgi-bin/*"
```

- Select proxy records requesting (a) particular site(s)

```
$ QDLOGSTATS HT_LOGS:*8080*.LOG /PATH="http://*.compaq.com*"
$ QDLOGSTATS HT_LOGS:*8080*.LOG /METHOD=POST /PATH="http://*sex*.*/*" /VIEW
```

- Records where the request was authenticated

```
$ QDLOGSTATS HT_LOGS:*.LOG /AUTHUSER=DANIEL
```

23.11 SECHAN Utility

The SECHAN utility (pronounced ''session'') is used by [INSTALL]SECURE.COM and its associated procedures to make file system security settings. It is also available for direct use by the site administrator. See SECHAN Utility.

23.12 Scrunch Utility *(obsolete)*

SCRUNCH Obsolete with 7.2

Changes with server-internal SSI document handling have made the SCRUNCH utility obsolete for WASD versions 7.2 and later. Previously SCRUNCHed documents will continue to be processed without needing to be explicitly UNSCRUNCHed.

23.13 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.

23.14 WASD Bench :^)

WASD Bench - an analogue to Apache Bench (Section 23.5) Why have it? Apache Bench 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 it's use.

```
$ WB -H
$ WB -C 10 -N 100 http://the.server.name/ht_root/exercise/0k.txt
$ WB -C 50 -N 500 -K http://the.server.name/ht_root/exercise/64k.txt
$ WB -C 10 -N 100 http://the.server.name/cgi-bin/cgi_symbols
```

WASD Bench also has an *exercise* option, functionality is not found in Apache Bench. It is basically to supercede similar functionality provided by the retired WWWRKOUT. The exercise functionality allows WASD Bench 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 WASD Bench 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 = "$HT_EXE:WB"
$ SPAWN/NOWAIT WB +e +s +n -n 100 -c  5    http://'p1'/ht_root/exercise/0k.txt
$ SPAWN/NOWAIT WB +e +s -k -n  50 -c  5 -k http://'p1'/ht_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'/ht_root/*.*
$ SPAWN/NOWAIT WB +e +s +n -n 100 -c  1    http://'p1'/ht_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'/ht_root/src/httpd/*.*
$!(etc.)
```

23.15 Server Workout *(obsolete)*

WWWRKOUT Obsolete with 8.0

As the WASD Bench :^) Utility now provides much of the *stress-test* functionality the WWWRKOUT utility supplied with earlier version of WASD has been declared obsolete.