WASD Hypertext Services - Technical Overview

5th September 1998

Version 5.2 release of the WASD VMS Hypertext Services HTTP server (September 1998).

Supercedes:

23rd July 1998 (v5.1)
30th March 1998 (v5.0)
15th November 1997 (v4.5)
1st October 1997 (v4.4)
1st August 1997 (v4.3)
1st July 1997 (v4.2)
1st June 1997 (v4.1)
1st October 1996 (v3.4)
1st December 1995 (initial *freeware* release, v3.1)
9th August 1995 (v2.3)
1st May 1995 (v2.1)

Please Note

As of July 1997 High Frequency Radar Division (HFRD) underwent a change of role and name. It is now Wide Area Surveillance Division (WASD). This package has been renamed in accordance with that.

Abstract

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

Also see “WASD Hypertext Environment” for information on using the WASD VMS Hypertext Services and “WASD Nuts and Bolts” for technical information on server design and implementation.

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
Senior Information Technology Officer
Wide Area Surveillance Division
Defence Science and Technology Organisation

Mark.Daniel@dsto.defence.gov.au

+61 (8) 82596031 (bus)
+61 (8) 82596673 (fax)

PO Box 1500
Salisbury
South Australia 5108

Printed Copy

This book may be available for printing to a PostScript printer. Use a browser to access a print menu in this same location.

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 © 1996-1998 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.

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.

Eric A. Young

This package *can* include cryptographic software (SSLeay) copyright by Eric Young (eric@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
```

MadGoat Software

For supporting non-Digital-TCP/IP (UCX) this software uses the NETLIB package by Matthew Madison:

```
permission is granted to copy and redistribute ...
for no commercial gain
```

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, **Digital TCP/IP Services for OpenVMS**, **VAX C**, **DEC C**, **VAX** and **AXP** are registered trademarks of Digital Equipment Corporation.

MultiNet is a registered trademark of Cisco Systems, Inc.

Pathway is a registered trademark of Attachmate, Inc.

TCPware is a registered trademark of Process Software Corporation.

Contents

Chapter 1 Introduction

Chapter 2 HyperText Transport Protocol Daemon - Overview

2.1 Server Features 2–1
2.2 Server Behaviour 2–2
2.3 TCP/IP Packages 2–3
2.4 International Features 2–4
2.5 HTTP Methods Usage 2–5
2.5.1 GET 2–5
2.5.2 POST & PUT 2–5
2.5.3 DELETE 2–7

Chapter 3 New to WASD? Start Here!

Chapter 4 Installation and Update

4.1 Installation DCL Procedure 4–3
4.2 Update DCL Procedure 4–4
4.3 Quick-Check 4–5
4.4 Re-Linking 4–5
4.5 Local Setup Suggestions 4–5
4.6 Organizing Documents 4–5
4.7 Reporting Problems 4–6

Chapter 5 Server Account and Environment

5.1 VMS Account 5–1
5.2 Account Support Files 5–2
5.3 HTTPd Command Line 5–5
5.3.1 Server Startup 5–5
5.3.2 Server Command Line Control 5–6
5.3.2.1 Accounting 5–7
5.3.2.2 Authentication 5–7
5.3.2.3 Cache 5–7
5.3.2.4 DCL/Scripting Subprocesses 5–7
5.3.2.5 DECnet Scripting Connections 5–8
5.3.2.6 Logging 5–8
5.3.2.7 Mapping 5–8
5.3.2.8 Shutdown and Restart 5–9
5.4 Multi-Homed Hosts With Digital TCP/IP 5–9

Chapter 6 Server Configuration

6.1 Content-Type Configuration 6–1
6.1.1 Explicitly Specifying Content-Type 6–2
6.2 Directives 6–3
6.3 Multi-Homed and Multi-Port Services 6–12
6.4 Access Logging 6–13
6.4.1 Log Format 6–13
6.4.2 Log Period 6–15
6.4.3 Log Per-Service 6–15
6.4.4 Log Naming 6–16

Chapter 7 Message Configuration

7.1 Behaviour 7–1
7.2 Message File Format 7–2
7.3 Supplied Message Files 7–3

Chapter 8 Mapping Rules

8.1 VMS File System Specifications 8–2
8.2 Rules 8–2
8.3 Rule Interpretation 8–5
8.4 Mapping Examples 8–5
8.5 Conditional Mapping 8–6

8.6 Mapping User Directories (*tilde* character (''~'')) 8–9

Chapter 9 Authentication and Authorization

9.1 Permissions, Path and User 9–3
9.2 Authorization Configuration File 9–3
9.3 Authorization Configuration Examples 9–6
9.3.1 KISS 9–7
9.4 Databases, Authentication and Group 9–8
9.5 Server Authorization Administration 9–9
9.6 SYSUAF-Authenticated Users 9–9
9.7 Controlling Server Write Access 9–12
9.8 User Password Modification 9–13

Chapter 10 Secure Sockets Layer

10.1 SSL Configuration 10–3
10.2 Certificate Management 10–5
10.3 SSLeay 10–6
10.4 SSL References 10–7

Chapter 11 Server Administration

11.1 Breaking-in To The Server! 11–2
11.2 HTTPd Server Reports 11–3
11.3 HTTPd Server Revise 11–6
11.4 HTTPd Server Action 11–7

Chapter 12 Scripting

12.1 Caution! 12–1
12.2 Scripting Environment 12–2
12.3 Script Run-Time 12–3
12.4 CGI Compliance 12–4
12.4.1 Example DCL Scripts 12–7
12.5 Raw HTTP Output 12–7
12.6 Raw HTTP Input 12–9
12.7 CGIplus Scripting 12–9
12.8 DECnet Scripting 12–13

12.8.1 Script System Environment 12–15
12.8.1.1 Proxy Access 12–15
12.8.1.2 DECnet Objects 12–16
12.8.1.3 Reducing Script Latency 12–17
12.8.1.4 DECnet/OSU Startup 12–17
12.8.2 CGI 12–18
12.8.3 OSU (DECthreads) Emulation 12–18
12.8.4 User Scripts 12–21
12.9 Java Scripts 12–22
12.10 HTTP Persistant-State Cookies 12–23

Chapter 13 Cache

13.1 Cache Suitability Considerations 13–2
13.2 Cache Content Validation 13–4
13.3 Cache Configuration 13–4
13.4 Cache Control 13–5

Chapter 14 Server Performance

14.1 File Record Format 14–3
14.2 Subprocess-based Scripting 14–4
14.3 DECnet-based Scripting 14–5
14.4 SSL 14–6
14.5 Suggestions 14–6

Chapter 15 HTTPd Web Update

Chapter 16 Utilities

16.1 Echo Facility 16–1
16.2 Where Facility 16–1
16.3 Xray Facility 16–2
16.4 StreamLF Utility 16–2
16.5 HTTPd Monitor 16–2
16.6 Server Workout *(stress-test)* 16–3

Chapter 1
Introduction

As of July 1997 High Frequency Radar Division (HFRD) underwent a change of role and name. It is now Wide Area Surveillance Division (WASD). This package has been renamed in accordance with that.

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 (June 1997 note: since first writing this the industry has settled on the term **intranet**). 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 VAX C and DEC C, within a DEC 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 on-line, on the Internet WWW).

Also see ‘‘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, internally available on-line (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. 1998 . . . I am reticent to make comments 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.

Chapter 2
HyperText Transport Protocol Daemon - Overview

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

2.1 Server Features

The package provides a complete implementation of a basic HTTP/1.0 server, including:

- concurrent, multi-threaded client support
- ''DELETE'', ''GET'', ''HEAD'', ''POST'' and ''PUT'' HTTP method support
- optional enhanced privacy using Secure Sockets Layer (SSL) technology (USA domestic, full-strength encryption)
- ''If-Modified-Since:'' / ''304 Not Modified'' functionality (document is only sent if modified since time specified by client)
- HTTP/1.0 de-facto persistent connections (i.e. ''Keep-Alive:'', reducing the number of TCP/IP connects required)
- versatile directory listing (generic and VMS-style)
- CGI-compliant subprocess scripting (with configurable, automatic, MIME content-type initiated activation)
- ''CGIplus'' scripting (offering reduced latency, increased throughput and reduced system impact when scripting)
- DECnet-based CGI scripting (with connection reuse)
- user account scripting (provided using the DECnet environment)
- 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 (extension)
- Server-Side Includes (HTML pre-processing)

- configurable cache, with time-based and forced re-validation (reload)
- clickable-image support (both NCSA and CERN formats)
- *smart* rule mapping (conditional rules)
- ‘‘Basic’’ and ‘‘Digest’’ user authentication and path/group-based authorization
- optional SYSUAF-authentication and VMS user security profile based file access control
- 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)
- host-level access control, on per-host or per-domain acceptance or rejection
- single server can support multi-homed (host name) and multiple port services
- customizable message database (capable of supporting non-English and concurrent, multiple languages)
- on-line server configuration, including reports on requests, loaded configuration, mapping rules, authorization information and graphical activity displays

2.2 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
   - processing of a clickable-image mapping file
   - directory listing
   - file/directory create/update
   - server administration
   - web file-system update
   - interpretation of a menu file
   - spawn a subprocess to to provide a CGI scripting environment:
     — standard CGI
     — WASD CGIplus

- 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 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.

2.3 TCP/IP Packages

The WASD server supports Digital TCP/IP Services (UCX) with a native image.

The ''UCX'' version is also known to work under TCPware v5.*n* (UCX compatibility emulation) from Process Software Corporation.

Using the excellent *freeware* NETLIB package from MadGoat Software (Matthew Madison), this server can also (potentially) support

- Cisco MultiNet for OpenVMS, any version
- PathWay from Attachmate Inc., any version
- TCPware from Process Software Corporation, any version
- CMU TCP/IP (VAX only) v6.5 or later **is not supported!** (too much variation from mainstream IP packages)

NETLIB requires VAX/VMS v5.2 or later, or OpenVMS Alpha v1.5 or later.

NETLIB must be obtained and installed separately. This is not a difficult undertaking for there is a familiar VMSINSTAL-driven installation and configuration.

The WASD NETLIB support was developed using v2.1, which may be obtained from

http://www.madgoat.com/netlib.html

ftp://ftp.madgoat.com/madgoat/

NETLIB support within the WASD server is tested in-house using the Digital TCP/IP Services package. Other TCP/IP packages may or may not perform as tested.

2.4 International Features

As of version 4.4 the HTTP server has undergone a limited degree of *internationalization*. The impetus for this development was the use of the package in some European countries where English is not the first language.

As this was not within the original critera while building the server it has meant an extensive rewrite of some functionality within the code (I hope it's appreciated ;^) The objective is to provide users of the server with a more familiar and therefore more comfortable environment. Unfortunately the server administrators are still confronted by an English-language interface and documentation, but then the package doesn't pretend to be *international software*!

Aspects

The international features only apply to the server, not to scripts!

- **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 7.

- **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.

- **Conditional Rule Mapping**

  Mapping rules map requested URL paths to physical or other paths (see Chapter 8). 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 example, requests originating with a language preference of ‘‘en, fr’’ from ‘‘.au’’ can receive an English version, whilst requests prefering ‘‘de, se, en’’ from ‘‘.de’’ and ‘‘.se’’ receive German and Swedish versions respectively.

For conditional mapping information see Section 8.5.

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 Chapter 12 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. As of v5.1 an installation DCL procedure simplifies the initial setup.

1. **Unzip Package**

   Install the files following the guidelines in Chapter 4.

2. **INSTALL Package**

   Server installation if performed using the INSTALL.COM DCL procedure, see Section 4.1.

   - **Link Package**

     Link the supplied object files to produce VMS executables for the system's version of VMS.

   - **Check Package**

     Using the installation procedure check the operation of the basic package as recommended in Section 4.3.

   - **Directory Protections**

     This section checks for correct directory protection in the HT_ROOT:[000000] tree and prompts to change it to WORLD:RE if it appears incorrect.

   - **Create Server Account**

     Using the installation procedure create the server account. For further information see Section 5.1.

   - **Disk Quota**

     Check if quotas are enabled on the target disk. Provide an ambit allocation for the server account. Review this at some stage.

   - **Copy Server Support and Configuration Files**

Using the installation procedure copy the example server support and configuration files as discussed in Section 5.2.

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. Information is provided in Section 5.2.
   2. The only configuration that should require immediate attention will be to the mapping rules, see Chapter 8.

   More generally server runtime configuration involves the considerations discussed in Section 4.6 along with the following aspects:

   - Configuring the HTTP server run-time characteristics, see Chapter 6.
   - Mapping request paths to the VMS file system, and to other things such as scripts, see Chapter 8.
   - Customizing some messages (or all if non-English language environment), see Chapter 7.
   - Establishing an authentication and authorization environment, see Chapter 9.

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 HT_ROOT:[LOG.SERVER]

   Remember, the basic installation's integrity can always be checked as described in Chapter 4, Section 4.3. 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 4.7.

Chapter 4
Installation and Update

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

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
```

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]
```

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]
```

In either case it is recommended to check the integrity of, then list the contents of, the archive before UNZIPping. This may be done as follows:

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

A complete installation/update (the former) 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
  16384  07-30-97  09:21   ht_root/vax/wwwrkout_netlib.exe
 ------                    -------
19109754                   996 files
```

An update (the latter) 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
  27136  07-30-97  09:21   axp/wwwrkout_netlib.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
```

The archive may then be UNZIPped. If this is a completely new installation a new directory tree will be created. If an update to an existing installation the tree will be created ‘‘over-the-top’’ of the existing structure.

When updating:

- 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.

- New files and directories are 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).

4.1 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 default to ‘‘no’’.

It performs the following tasks:

1. **Link Executables** - Links the package object code to produce images for the local version of VMS.
2. **Server Quick-Check** - Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package. See Section 4.3.
3. **Directory Protections** - This section checks for correct directory protection in the HT_ROOT:[000000] tree and prompts to change it to WORLD:RE if it appears incorrect.
4. **Create Server Account** - Creates an unprivileged account for the HTTP server to execute within. Sets permissions on files and directories allowing appropriate access for this account.
5. **Disk Quota** - Check if quotas are enabled on the target disk. Provide an ambit allocation for the server account. Review this at some stage.
6. **Server Support/Configuration Files** - Copies example HTTP server configuration and support files from the [EXAMPLE] directory to the [HTTP$SERVER] and [LOCAL] directories. No copy is made without prompting for permission.

After UNZIPping the package do the following:

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

NOTE: The installation procedure provides a reasonable, *vanilla*, single-node set-up. If something different is required a little time and thought may be required further configuring startup, etc.

Support files to consider when customizing startup, etc. (see Section 5.2 for further detail):

- STARTUP.COM

- STARTUP_LOCAL.COM
- HTTPD_BATCH.COM
- HTTPD80.COM
- LOGIN.COM

4.2 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 default to ‘‘no’’.

It provides the following functions:

1. **Link Executables** - Links the package object code to produce images for the local version of VMS.
2. **Server Quick-Check** - Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package. See Section 4.3.
3. **Server Support/Configuration Files** - Copies changed example HTTP server configuration and support files from the [EXAMPLE] directory to the [HTTP$SERVER] and [LOCAL] directories. This copy is made without prompting for permission.
4. **Post-Update Cleanup** - Prompts for permission to execute the post-update procedure described below.
5. **Purge Files** - Prompts for permission to purge the entire HT_ROOT:[000000...] tree.

After UNZIPping the updated package do the following:

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

Post-Update

When confident regarding the success of the update (these activities can be part of the update DCL procedure).

- The procedure UPDATEAFTER.COM should be executed. This will remove files known to exist from previous versions and relocated or no longer required.

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

- The directory tree should be purged.

4.3 Quick-Check

Once installed or updated it is possible to check the basic package at any time using the FREEWARE_DEMO.COM procedure. It should detect the TCP/IP agent in use (if not specify UCX or NETLIB as a parameter). Follow the displayed instructions.

```
@device:[HT_ROOT]FREEWARE_DEMO
```

Problem? SYSTEM-F-SHRIDMISMAT? See next section!

4.4 Re-Linking

After an 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 4.2.

4.5 Local Setup Suggestions

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

- HT_ROOT:[HTTP$SERVER]
- HT_ROOT:[LOCAL]
- HT_ROOT:[SCRIPT_LOCAL]

To prevent the overwriting of local configuration files it is suggested these be placed in the HT_ROOT:[LOCAL] directory. Startup files will most probably be placed where-ever the local site manages system startup. These could be placed in the HT_ROOT:[LOCAL] directory if there is nowhere more appropriate. Authentication databases could also be placed in the [LOCAL] directory. The server account's support files (LOGIN.COM, etc.) may be copied from the HT_ROOT:[EXAMPLE] directory into HT_ROOT:[HTTP$SERVER] and customized as necessary.

In like manner, local DCL scripts or DCL *wrappers* for executable scripts should be placed in HT_ROOT:[SCRIPT_LOCAL]. Executables should continue to be placed in HT_EXE:

Changes that are made in package configuration files, etc., can then be propagated to the local areas as appropriate.

4.6 Organizing Documents

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 in something like **WEB:[000000]**. This logical device could be provided with the following DCL introduced into the server startup procedures:

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

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 (see Chapter 8).

WASD has a single section of the file system for the core Web files, such as the Division home page, help areas, Web documentation, etc., it can be accessed using the logical device WEB:[000000]. Incorporated with this are other subdirectories providing storage for specific collections of data, such as committee minutes, administration announcements, etc.

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).

Reasons for an area being physically integrated with the core Web data area can be legitimate (e.g. there is really nowhere else it reasonably belongs), convenience (e.g. let's quickly put it here) or by logical necessity (it really does below as part of the core Web enviroment, e.g. documentation on HTML, etc.)

Guidelines

In general, only Web-related files need to go into the core physical Web file area. All other groupings should, if possible, be decentralised into the portion of the file system they represent and logically placed in the server's path using rules in the mapping file. That is, a given project's Web files should be located in the project's part of the file system. If it doesn't have any then it may be a candidate for location in the core area.

When locating a Web area in a physically distinct area it is possible the maintainers of that data will already have the correct access controls. If locating an area in the core hypertext environment it will be necessary to give the mantainer ownership of the directory area (and possibly disk quota), or provide ACL access if multiple maintains are involved.

When locating Web-accessable data in a physically distinct area it will be necessary to update the mapping file with a new rule (see Chapter 8). If located within the core Web area the rules do not need to be adjusted.

4.7 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 a 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.

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@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.

5.1 VMS Account

NOTE: 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.

A general rule is more is better, after all, it will only use as much as it needs!

The following provides a guide to the account:

```
Username: HTTP$SERVER                       Owner:  HyperText Daemon
Account:  HTTPD                             UIC:    [377,377] ([HTTPD,HTTP$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   00000000001111111112222  Secondary 00000000001111111112222
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:       300000
Maxacctjobs:      0  Shrfillm:      0  Pbytlm:           0
Maxdetach:        0  BIOlm:       512  JTquota:       1024
Prclm:          100  DIOlm:       512  WSdef:         1000
Prio:             4  ASTlm:       600  WSquo:         2000
Queprio:          0  TQElm:       100  WSextent:     20000
CPU:        (none)  Enqlm:       256  Pgflquo:     200000
Authorized Privileges:
  NETMBX    TMPMBX
Default Privileges:
  NETMBX    TMPMBX
```

5.2 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 are built by the package.

- **HTTPD.EXE** - This provides native support for Digital TCP/IP services (UCX).
- **HTTPD_NETLIB.EXE** - This provides NETLIB support for packages other than UCX.

Two additional executables are built by the SSL package (see Chapter 10).

- **HTTPD_SSL.EXE** - This provides SSL for Digital TCP/IP services (UCX).
- **HTTPD_SSL_NETLIB.EXE** - This provides SSL based on NETLIB, for packages other than UCX.

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 NETMBX, TMPMBX, PRMMBX, PSWAPM, SYSNAM and SYSPRV 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,PRMMBX,PSWAPM,SYSPRV,SYSNAM)
```

Logical Names

The following logical names are essential for the operation of the HTTPd server and must be defined before startup:

- **HTTPD$AUTH -** location of the authentication/authorization configuration file (defined system-wide, or in the job table if server-specific)
- **HTTPD$CONFIG -** location of the configuration file (defined system-wide, or in the job table if server-specific)
- **HTTPD$MAP -** location of the mapping rule file (defined system-wide, or in the job table if server-specific)
- **HTTPD$MSG -** location of the message file (defined system-wide, or in the job table if server-specific)
- **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 required server certificate.
- **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.
- **HT_AUTH -** directory containing authentication/authorization databases (files)
- **HT_EXE -** directory containing the executable images
- **HT_LOGS -** optional definition, for convenient log file specification

- **HT_SERVER_LOGS -** optional definition, for convenient detached server process log file specification

The following logical name is created by the executing HTTP server and defines the name of the control mailbox:

- **HTTPD*port*$CONTROL**

The following logical names are created by the executing HTTP server if the HTTPd monitor utility is enabled:

- **HTTPD*port*$COUNT1**
- **HTTPD*port*$COUNT2**
- **HTTPD*port*$PID**
- **HTTPD*port*$REQUEST**

Server Process Logging Directory

The server process log directory (output for the detached HTTP server processes) may require explicit access controls for the HTTPd account. This can be done by applying an ACL similar to the following:

```
$ SET SECURITY HT_ROOT:[LOG]SERVER.DIR -
  /ACL=((IDENT=HTTP$SERVER,ACCESS=R+W+E, OPTIONS=DEFAULT), -
        (IDENT=HTTP$SERVER,ACCESS=R+W+E), -
        (IDENT=*,ACCESS=NONE, OPTIONS=DEFAULT), -
        (IDENT=*,ACCESS=NONE))
```

As with the ACL on the server executable this can be done once, at installation (or, if right over the top, at each server startup). Appropriate disk quotas may also need to be applied. Also see Section 5.3.2.6.

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.

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

HTTPD_BATCH.COM

This procedure is SUBMITted by the server STARTUP.COM using the HTTP server account, and simply creates a detached process under the control of HTTPD80.COM, or its equivalent. A parameter may be provided to this procedure. This roughly represents the port number of a server but actually represents a file HTTPD'number'.COM which controls a specific server, see below.

See HT_ROOT:[EXAMPLE]HTTPD_BATCH.COM

HTTPD80.COM

This procedure is suggested for controlling the execution of the HTTPd server and it's image. It is run in the detached process created by HTTPD_BATCH.COM. Multiple instances of this file may be created, each having a unique number, for controlling multiple servers, either on an individual node or within a cluster. Actually, just any unique string may be used for indentification.

See HT_ROOT:[EXAMPLE]HTTPD80.COM

LOGIN.COM

This procedure provides the HTTP server account login control.

See HT_ROOT:[EXAMPLE]LOGIN.COM

5.3 HTTPd Command Line

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

5.3.1 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.

- **/ACCEPT=** comma-separated list of hosts/domains allowed to connect to the server
- **/AUTHORIZATION=SSL** the ''SSL'' parameter causes all authentication (both SYSUAF and HTA database) to be available only via ''https:'' requests (see Chapter 10)
- **/CGI_PREFIX=** the prefix to the CGI symbol names created for a script (defaults to ''WWW_'', similar to the CERN VMS HTTPd, see Chapter 12)
- **/DO=** command to be performed by the executing server (see Section 5.3.2)
- **/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]
- **/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$OUTPUT'' the server issues log entries to <stdout>, allowing user-defined log formats to be easily checked and refined.
- **/NETBUF=** minimum number of bytes in the network read buffer
- **/OUTBUF=** number of bytes in the output buffer (for direct file transfers, buffered output from menu interpretation, HTML-preprocessing, etc.)
- **/PERIOD=** overrides the configuration parameter [LogPeriod]
- **/PORT=** overrides the configuration parameter [Port] BUT is in turn overridden by the [Service] configuration parameter and /SERVICE= qualifier (is really only useful for use with the /DO= qualifier).
- **/PRIORITY=** server process priority (default is 4)
- **/[NO]PROFILE** allows SYSUAF-authenticated username security profiles to be used for file access (see Section 9.6
- **/PROMISCUOUS[=password]** server will accept any authentication username/password pair (used for testing, demonstrations, etc.)
- **/SERVICE=** comma-separated, list of server services (overrides the [Service] configuration parameter)
- **/[NO]SSL[=version]** controls Secure Sockets Layer protocol behaviour. The version can be any of ''2'', ''3'' or ''23'' (i.e. both 2 and 3, and the default) specifying which SSL protocol version the server will service.
- **/SUBBUF=** number bytes in a subprocess' SYS$OUTPUT buffer
- **/[NO]SWAP** controls whether the server process may be swapped out of the balance set (default is swapping disabled)
- **/[NO]SYSUAF[=ID,SSL]** allows or disallows (D) username authentication using the server system's SYSUAF (see Section 9.6, the optional ''SSL'' parameter causes SYSUAF authentication to be available only via ''https:'' requests (see Chapter 10), and the optional ''ID'' parameter makes SYSUAF authentication only available to account possessing a specific identifier. See Section 9.6.
- **/VERSION** merely displays the executable's version string and the copyright notice

Note: buffer sizes apply on a per-request (thread) basis, and may be tailored for specific environments at server startup.

5.3.2 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_NETLIB"
```

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, or multiple servers on the one system are being used, then it will be necessary to provide a ''/PORT='' qualifier with any command, and possibly issue it multiple times. Of course, these commands are available from batch jobs as well as interactively.

5.3.2.1 Accounting

Server counters may be zeroed. These counters are those visible from the *statistics* administration menu item and when using the HTTPDMON utility.

```
$ HTTPD /DO=ZERO
```

5.3.2.2 Authentication

See Chapter 9.

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.

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

5.3.2.3 Cache

Server cache control may also be exercised from the server administration menu, see Chapter 11. The file cache (see Chapter 13) 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
```

5.3.2.4 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 Chapter 12.

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
```

5.3.2.5 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
```

5.3.2.6 Logging

Server logging control may also be exercised from the server administration menu, see Chapter 11.

Open a log file.

```
$ HTTPD /DO=LOG=OPEN
```

Optionally open and specify a log file name. This overrides any /LOG=*file-name* or HTTPD$LOG specified file name.

```
$ HTTPD /DO=LOG=OPEN=file-name
```

The log file may be closed and reopened using a different name (allowing continuous operation but, for example, with daily log files). This overrides any /LOG=*file-name* or HTTPD$LOG specified file name.

```
$ HTTPD /DO=LOG=REOPEN=file-name
```

Close the log file.

```
$ HTTPD /DO=LOG=CLOSE
```

Unwritten log records may be flushed to the file.

```
$ HTTPD /DO=LOG=FLUSH
```

The format and period specification of the log may be changed (only takes effect after the next log file open/reopen) using

```
$ HTTPD /DO=LOG=FORMAT=string
$ HTTPD /DO=LOG=PERIOD=string
```

5.3.2.7 Mapping

See Chapter 8.

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

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

5.3.2.8 Shutdown and Restart

Server shutdown may also be exercised from the server administration menu, see Chapter 11.

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
```

This variant restarts the server immediately regardless of existing connections.

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

5.4 Multi-Homed Hosts With Digital TCP/IP

A multi-homed host responds to connections for more than one network address. Details on configuring Digital TCP/IP (UCX) for this behaviour may be found in the *Digital TCP/IP Management* manual, section on ‘‘Subnetwork Routing’’ (3.4.8 in the November 1995 edition) and *Digital TCP/IP Management Command Reference*, section on ‘‘SET INTERFACE’’ (November 1995 edition). The server configuration parameter [Service] allows for the processing of requests for multiple host-names and/or ports (see Section 6.2) while conditional mapping rules enhance this basic capability (see Section 8.5).

Assuming an alternate, unused host name exists (for example):

```
$ UCX
UCX> SET HOST "second.host.name" /ADDRESS=131.185.30.2 /ALIAS=("second","SECOND")
UCX> [EXIT]
$
```

The system IP interface can be determined as follows (in this example it is SE0, the LO0 is the localhost interface):

```
UCX> SHOW INTERFACE
                                                          Packets
Interface   IP_Addr         Network mask          Receive          Send     MTU

 SE0        131.185.30.1    255.255.0.0            595567        262764    1500
 LO0        127.0.0.1       255.0.0.0                   1           573   65535  ?
UCX> SHOW INTERFACE SE0 /FULL
Interface: SE0
  IP_Addr: 131.185.30.1      NETWRK: 255.255.0.0       BRDCST: 131.185.255.255
                      Ethernet_Addr: AA-00-04-00-59-04    MTU:  1500
.
.
.
  Restarting attempts                  0
  Successful restarts                  0
```

Note the network and broadcast masks of the interface above. The basic procedure then is to create a pseudo-interface (the ‘‘A’’ added to the hardware device name) as follows:

```
UCX> SET INTERFACE SEA0 /HOST="second" /NET=255.255.0.0 /BROAD=131.185.255.255
UCX> SET PROTOCOL IP /FORWARD
```

After the above commands the interfaces should appear as shown below:

```
UCX> SHOW INTERFACE
                                                          Packets
Interface   IP_Addr         Network mask          Receive          Send     MTU

 SE0        131.185.30.1    255.255.0.0            595567        262764    1500
 SEA0       131.185.30.2    255.255.0.0               169            99    1500
 LO0        127.0.0.1       255.0.0.0                   0          2263   65535  ?
```

These commands can be done interactively, during each startup, or placed in the permanent UCX configuration database once, as follows:

```
UCX> SET CONFIG INTERFACE SEA0 /HOST="second" /NET=255.255.0.0 /BROAD=131.185.255.255
UCX> SET CONFIG PROTOCOL IP /FORWARD
```

Note: Originating Host Address

Digital TCP/IP Services v4.1 at least, appears to determine an outgoing connection’s originating host address from the IP address of the ‘‘highest’’ pseudo-interface defined (in this case from the SEA0, resulting in an apparent source address of 131.185.30.2). This can cause some confusion on remote hosts as to the origin of a connection, and otherwise be generally undesirable.

A solution appears to be to assign the primary address to the last defined interface. This means the original interface must be removed using a

```
UCX> SET CONFIG NOINTERFACE SE0
```

then explicitly redefining it and then the new interface, ensuring the first is the non-primary address and the second the primary.

Chapter 6
Server Configuration

WASD's HTTPd configuration was originally a subset based on the CERN HTTPd. It has now evolved considerably. 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 11 for further information.

6.1 Content-Type Configuration

HTTP uses a quasi-standard 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.

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

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). When this is not done a directory listing shows ‘‘[?]’’ in place of an icon.

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.

If a file type is not recognised (i.e. no [AddType] 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.

6.1.1 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 hypertext 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:

online hypertext demonstration

6.2 Directives

NOTE: The format of the configuration file has been changed between version 3 and version 4. This was to support on-line server configuration and administration (see Chapter 11). The version 3 format will still be read by the HTTPd but the first time the configuration is updated on-line it is saved in the version 4 format, described below.

The example configuration file for the WASD HTTP server can be viewed.

online hypertext link

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.

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 numeric 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 menu (see Section 11.2). 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***
   [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
[AddDirIcon]      /icon/-/dir.gif      [DIR]
[AddParentIcon]   /icon/-/back.gif     [<--]
[AddUnknownIcon]  /icon/-/unknown.gif  [???]
```

5. **[AddType]** ***.suffix representation encoding 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. The description is used as documentation for directory listings.

   Examples:

```
[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
```

   (Versions prior to 3.2 used this configuration directive for the MIME content-type to determine whether a file was transfered record-by-record or in binary. This is no longer required and should be removed from existing configuration files.)

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

   Enables or disables BASIC username authentication. See Chapter 9.

7. **[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).

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

   Enables or disables Digest username authentication. See Chapter 9.

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

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

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

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

11. **[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.

12. **[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.

13. **[AuthVMS]**

    (Retired in v4.4, server /SYSUAF qualifier provides this)

14. **[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.

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

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

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

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

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

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

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

19. **[CacheFrequentHits]** ***integer*** *(default: 0)*

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

20. **[CacheFrequentSeconds]** ***integer*** *(default: 0)*

If a file has been hit at least [CacheFrequentHits] times in total and the last was within the number of seconds here specified it will not be a candidate for reuse. See Chapter 13.

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

    Size of the hash table.

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

    Maximum memory allocated to the cache, in kilobytes.

23. **[CacheValidateSeconds]** ***integer*** *(default: 0)*

    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.

24. **[CommentedInfo]** ENABLED|DISABLED *(default: DISABLED)*

    Includes, as META information, the software ID of the server and any relevant VMS file specification for directory listings, etc.

25. **[DclCgiPlusLifeTime]** ***integer*** *(default: 0)*

    This value represents time in minutes. 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 number of minutes after it last processed a request. This helps prevent sporadically used scripts from clogging up a system.

26. **[DclFullRequest]** ENABLED|DISABLED *(default: DISABLED)*

    Versions of the server prior to 4.3 supplied the full request (header then body) to the script. This was not fully CGI-compliant. Versions 4.3 and following supply only the body, although the previous behaviour may be explicitly selected by enabling this configuration parameter.

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

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

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

    One or more file type (extension) specification and scripting verb pairs. See Section 12.3.

29. **[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''.

30. **[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.

31. **[DclZombieLifeTime]** ***integer*** *(default: 0)*

    This value represents time in minutes. If this value is zero the use of persistant DCL subprocesses is disabled. If non-zero the *zombie* subprocess is terminated the specified number of minutes after it last processed a request. This helps prevent zombie processes from clogging up a system. See Chapter 12.

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

    The number of minutes a DECnet scripting connection is maintained with the network task. Zero disables connection reuse.

33. **[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.

34. **[DirAccess]** ON|OFF| 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.

35. **[DirDescription]** ON|OFF| SELECTIVE *(default: DISABLED)*

    Retired in v5.0, replaced by [DirDescriptionLines]

36. **[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).

37. **[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)
- **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.

38. **[DirNoImpliedWildcard]** ON|OFF *(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.

39. **[DirNoPrivIgnore]** ON|OFF *(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.

40. **[DirOwner]** ON|OFF *(default: DISABLED)*

    Allows specification and display of the RMS file owner information.

41. **[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
```

42. **[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.

43. **[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
```

44. **[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.

45. **[DNSLookup]** 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 numeric internet address. If not enabled any [Accept], [Reject] directive, etc., must be expressed as numeric addresses.

This functionality significantly slows request processing. See Chapter 14.

NOTE: From version 4.4 the HTTPd processes initial client connection at AST delivery level. Some NETLIB-supported TCP/IP packages cannot perform host name resolution at this level and will ‘‘hang’’. If using NETLIB with [DNSlookup] enabled and this seems to occur disable it (NETLIB DNSlookup certainly does work with Digital TCP/IP).

46. **[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)
```

47. **[ErrorSourceInfo]** ENABLED|DISABLED *(default: DISABLED)*

    When an error message is generated META information is included showing the server version with source code module and line reporting the error. This is useful information during development and often during general use. Only disable if this concerns you.

48. **[ErrorInfo]**

    (Retired in v4.4, message configuration provides this)

49. **[ErrorSysAdmin]**

    (Retired in v4.4, message configuration provides this)

50. **[InputTimeout]** ***integer*** *(default: 2)*

    Number of minutes to allow a connection request to be in progress without submitting a complete request header before terminating it.

51. **[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 (see Logical Names.

52. **[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 numeric 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.*
    ```

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

    Specifies one of three pre-defined formats, or a user-definable format. See Section 6.4.1.

54. **[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 6.4.4.

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

    Specifies a period at which the log file is changed. See Section 6.4.2.

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

    When multiple services are specified (see Section 6.3) a separate log file will be created for each if this is enabled. See Section 6.4.3.

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

    Allows monitoring via the HTTPDMON utility (see Section 16.5. Adds slight request processing overhead.

58. **[OutputTimeout]** ***integer*** *(default: 10)*

    Number of minutes to allow a request to be output before terminating it.

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

    IP port number for server to bind to.

60. **[PutMaxKBytes]** ***integer*** *(default: 250)*

    Maximum size of an HTTP POST or PUT method request in Kilobytes.

61. **[PutVersionLimit]** ***integer*** *(default: 3)*

    File created using the POST or PUT methods have the specified version limit applied.

62. **[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 numeric 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.*
    ```

63. **[RequestHistory]** ***integer*** *(default: 0)*

    The server can keep a list of the most recent requests accessable from the server administration menu. 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.

64. **[Search]** ***path*** *(no default)*

    Specifies the **physical path** to the default query-string keyword search script.

    Examples:

    ```
    Search /ht_root/script/query
    ```

65. **[Service]** ***string*** *(no default)*

    This parameter allows SSL, multi-homed hosts and multiple port serving to be specified, see Section 10.1 and Section 6.3.

66. **[SSI]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables Server Side Includes (HTML pre-processing).

67. **[SSIaccesses]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables Server Side Includes (HTML pre-processing) file access counter.

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

    Enables or disables Server Side Includes (HTML pre-processing) DCL execution functionality.

69. **[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 the conversion. Also see [StreamLFpaths].

70. **[StreamLFpaths]** ***string*** *(no default)*

    Specifies which path(s) automatic conversion of VARIABLE record format files to STREAM-LF is allowed. If no path is provided all conversion is disabled. To allow conversion on all paths use ''/*''. Also see [StreamLF].

    Example:

```
[StreamLFpaths]
/ht_root/*
/web/*
```

71. **[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.

    Examples:

```
[Welcome]
home.html
home.htm
home.menu
home.mnu
```

6.3 Multi-Homed and Multi-Port Services

By default the server is defined by the local host name and port specified by either [port] or /PORT. The WASD server is capable of concurrently supporting a number of services on different port numbers and, if the system is configured to be multi-homed (see Section 5.4), more than one host name. Using the [Service] 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.

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

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

This capability combined with conditional rule mapping (see Section 8.5) offers versatile and powerful service provision. The first port specified in the service list becomes the ''official'' port of the server, using the local host name.

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

6.4.1 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.

1. **COMMON -** This is the most common, base logging format for Web servers. COMMON is the default log format.
2. **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).
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

- **AR -** authentication realm (if any)
- **AU -** authenticated user name (if any)
- **BB -** bytes in body (excludes response header)
- **BY -** bytes in response (includes header)
- **CN -** client host name (or address if DNS lookup disabled)
- **CA -** client address
- **EM -** request elapsed time in milliseconds
- **ES -** request elapsed time in fractional seconds
- **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

- **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.

Note on “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
```

6.4.2 Log 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 [LogFileName] 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.

6.4.3 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 (see Section 6.3).

6.4.4 Log Naming

When per-period or per-service logging is enabled the access log file has a 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 dot-numeric address is used, hyphens being substituted for the periods. Accepted values for the [LogNaming] configuration directive are:

- NAME (default)
- ADDRESS

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

```
HT_LOGS:WASD_80_ACCESS.LOG
HT_LOGS:131-185-250-202_80_ACCESS.LOG
```

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

```
HT_LOGS:WASD_80_19971013_ACCESS.LOG
HT_LOGS:131-185-250-202_80_19971013_ACCESS.LOG
```

Chapter 7
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.

7.1 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!

7.2 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 :^)

Care must be taken with the message file or the server may refuse to start

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] 4.4
[language] 1 se *.se
[language] 2 de *.de,*.su
[language] 3 en

[general]

en 01 Sanity check failure.
de 01 (the same message but in German, which I cannot speak :^)
se 01 (the same message but in Swedish, which I also cannot speak :^)
en 02 String overflow.
de 02 (the same message but in German, which I cannot speak :^)
se 01 (the same message but in Swedish, \
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.

1. **[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.

2. **[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.) 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. Setting the language's ordering number to zero disables the language completely. All messages associated with it will then be ignored.

3. **[group-name] -** The messages are divided into groupings to make them easier to manage. Each group begins with the group name directive.

4. **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.

7.3 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 8
Mapping Rules

Mapping rules are used in four primary ways.

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.

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.

Hence, path authorization is always applied to the path supplied with the request, not to the results of any rule mapping that may have occured! This means authorization paths may be administered without any ambiguity introduced by any rule mapping that may occur.

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.

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.

The server administration rule mapping facility allows arbitrary paths to be checked against the rule database in real-time. See Section 11.2.

Any changes to the mapping file may be (re)loaded into the running HTTPd server using the following command on the server system:

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

Also see Chapter 6 for daemon configuration.

Server Mapping Rules

A server's currently loaded mapping rules may be interrogated. See Chapter 11 for further information.

8.1 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 assumed to be 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 the MFD is completely optional, 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.]
```

8.2 Rules

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.

   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 ''/'').

REDIRECT Rule

1. **redirect** ***template result***

   If the URL path matches the template, substitute the *result* string for the path. Process no further rules. The *result* must be a full URL (*http://host/path*), and is used to redirect requests to another server on a separate host.

EXEC and SCRIPT, Script Mapping Rules

EXEC+ and SCRIPT+

Also see Chapter 12.

The ''EXEC'' and ''SCRIPT'' directives have the variants ''EXEC+'' and ''SCRIPT+''. The variants behave in exactly the same fashion and simply mark the rule as representing a CGIplus script environment. See Section 12.7. **Caution!** If changing rules involving these variants it is advised to restart the server rather than reload. Some conflict is possible when using new rules while existing CGIplus scripts are executing.

The ''EXEC'' rules maps CGI script **directories**.

The ''SCRIPT'' rules maps CGI script **file names**. It is a little different to the ''EXEC'' rule and an extension to the CERN rules.

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. **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/*
   ```

8.3 Rule Interpretation

The rules are scanned from first towards last, until a *pass*, *exec*, *script* or *fail* is encountered, when processing ceases and final substitution occurs. *Map*ped rules substitute the *template* with the *result* and continue to the next rule.

Use of wildcards in *template* and *result*:

- The **template** may contain one or more asterisk (''*'') wildcard symbols. These match zero or more characters up until the character following the wildcard (or end-of-string). If no wildcard is present then the path must match the *template* exactly.
- The **result** may contain one or more asterisk (''*'') wildcard symbols. It must not contain more wildcards than the *template*. The *result* wildcards are expanded to replace the matching characters of the respective *template* wildcards. 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.

8.4 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

The *result* string of these rules should correspond to to a VMS physical file path.

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. See Chapter 12 for other information on scripting.

   ```
   exec /cgi-bin/* /cgi-bin/*
   ```

2. 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. See Chapter 12 for other information on scripting.

   ```
   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/*
   ```

8.5 Conditional Mapping

The purpose of *conditional mapping* is to **apply** rules only after certain criteria other than the initial path match are met. These criteria serve to create conditional mapping rules, and were introduced in version 4.4.

THIS OFFERS A POWERFUL TOOL TO THE SERVER ADMINISTRATOR!

Conditional mapping can be applied on the following criteria:

- authenticated remote user
- client internet address
- browser-accepted languages

- browser-accepted character sets
- browser-accepted content-types
- browser identification string
- host and port specified in request header
- HTTP method (GET, POST, etc.)
- proxy/gateway host(s) request forwarded by
- request scheme (protocol . . . ‘‘http:’’ or ‘‘https:’’)
- server name
- server port

Note that path authorization is always applied to the path supplied with the request, not to the results of any rule mapping that may have occured! This means authorization paths may be administered without any ambiguity being implied by any rule mapping that may occur.

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.

Matching is done by simple, case-insensitive, string comparison, using the wildcards ‘‘*’’, matching one or more characters, and ‘‘%’’, matching any single character.

White-space (spaces and TABs), wildcards and the delimiting ‘‘[’’ and ‘‘]’’, are forbidden characters and cannot be used within condition matching strings, nor can they be encoded for inclusion in any way (for simplicity and speed of processing). These characters are uncommon in the information being matched against, but if one does occur then ‘‘match’’ it using a single character wildcard (‘‘%’’).

While conditionals are powerful adjuncts to *smart* serving they do add significant overhead to rule mapping and should be used with this in mind.

Conditionals

- **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)
- **fo: -** request forwarded by proxy/gateway host(s) (‘‘Forwarded:’’ request header field)
- **hh: -** host and port request directed to (‘‘Host:’’ request header field)
- **ho: -** browser host internet name or address

- **me: -** request HTTP method
- **sc: -** request *scheme* (protocol), ‘‘http’’, and if SSL is in use ‘‘https’’ (see Chapter 10)
- **sn: -** server name
- **sp: -** server port
- **ru: -** authenticated remote user name
- **ua: -** browser (‘‘User-Agent:’’ 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 9).

```
pass /ht_root/local/* /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]
```

Note that rule processing for any particular path may be checked using the server administration menu.

8.6 Mapping User Directories (*tilde* character ("~"))

This server will map user directories using the same mechanisms as for any other. No reference is made to SYSUAF.DAT, user support is 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. As the naming of home directories using the username is a common practice this mechanism should suffice in the majority of cases. Where there is no such correspondance individual rules could be used for each user. User scripts can also be supported using WASD's DECnet scripting environment. See Section 12.8.4 for further detail.

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 9
Authentication and Authorization

Authentication is the verification of a user's identity, usually through a username/password combination. **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 9.3.1.

Overview

Server authorization is performed using a configuration file, authentication database and optional authorization grouping database, 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 databases.

AUTHORIZATION IS ALWAYS APPLIED TO THE PATH SUPPLIED WITH THE REQUEST, not to the results of any rule mapping that may have occured! This means authorization paths may be administered without ambiguity. With requests containing a script, 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.

For the WASD HTTPd user authentication can be done via the server system's **SYSUAF database** (**not recommended except except when employing SSL or in the most secure of LAN environments**), see Section 9.6 for further information.

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

The **authorization database** can be refered to as the *group*, and refers to a collection of usernames and associated *permissions*.

Separate databases are not always necessary, permission information can be stored in the same database as the password information. Paths are generally authorized on a per-group basis. Path access can be also be restricted on an internet host name/domain, internet address and/or authenticated username basis. If only a small number of usernames are to be associated with a path it is possible to list them all against the path and have permissions derived directly from the authentication database (realm).

Authentication Policy

A *policy* regarding when authorization can be used may be established for the server. This can restrict authentication challenges to ''https:'' (SSL) requests (see Chapter 10), thereby ensuring that the authorization environment is not compromised by use in non-encrypted transactions. Two HTTPd qualifiers provide this.

- **/AUTHORIZATION=SSL** restricts **all** authentication/authorization transactions to the SSL environment.
- **/SYSUAF=SSL** restricts only SYSUAF authenticated transactions to the SSL environment.

Note also that individual paths may be restricted to SSL requests using either the mapping conditional rule configuration or the authorization configuration files. See Conditionals and Access Restriction List.

In addition two configuration parameters have a direct role in an established authorization policy.

- **[AuthFailureLimit]** sets the maximum unsuccessful attempts allowed before a username is then always refused authentication. After this limit is reached the cache must be purged to re-allow the user access, see Section 5.3.2.2.
- **[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.

Authorization Scheme

The term *authorization scheme* refers to the HTTP method of authenticating the user. The WASD HTTPd can employ one or other, or both of the ''Basic'' and ''Digest'' schemes. Both schemes use a username/password mechanism, with Digest being preferable because the password is transmitted encrypted, and with Basic only encoded (and therefore easily readable).

Note

The Digest scheme has, to date, only been tested against NCSA X Mosaic 2.7-4b, which seems to behave a little *flakey* when reloading documents, and does not elegantly support stale nonces.

9.1 Permissions, Path and User

Both paths and usernames have permissions associated with them. A path may be specified as read-only, read and write, or write-only (yes, I'm sure someone will want this!) 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.

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
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

9.2 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.

Configuration directives begin either with a ‘‘[realm]’’ or ‘‘[realm;group]’’ specification, or with the forward-slash of a path specification. 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] specification, introducing a new authentication grouping. Within these brackets is specified the realm (and database) name, and then an optional group (and database) name separated by a semi-colon. All path specifications following this are authenticated against the specified realm database, and permissions obtained from the group database (or authentication database if group not specified), until the next [realm;group] specification.

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

  ```
  [authentication-database]
  ```

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

  ```
  [authentication-database ; authorization-database]
  ```

- **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
  ```

The same path cannot be specified against two different realms. 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.

Reserved Realms

The following realm names (authentication database names) are reserved and have special functionality.

1. **EXTERNAL -** Any authentication and authorization will be done in some way by an external CGI script. None is attempted by the server.
2. **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.
3. **VMS -** Use the VMS 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 (see Chapter 10) 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.

   See Section 9.6 for further information.

Reserved Username

The following username is reserved.

1. **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 numeric 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.
- **Request Scheme -** (protocol) either ''http:'' or secured via ''https:'' (SSL)

- **User Names -** are indicated by a leading circumflex, 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.

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

9.3 Authorization Configuration Examples

1. In the following example the authentication realm is ‘‘WASD’’ and the permissions group ‘‘SOCIALCLUB’’. The directive allows those authenticated from the WASD realm and in the SOCIALCLUB group to read and write, and the world to read.

```
[WASD;SOCIALCLUB]
/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;SOCIALCLUB]
/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. In the following example the authentication realm and group are a single 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]
/web/everyone/* get,post;get
/web/select/few/* get,post
```

4. 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.

```
[COMPANY]
/web/docs/* ~Howard,~George,~Fred,r+w ; r
/web/accounts/* ~George,r+w ; r
```

5. 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 get read access.

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

6. 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.

```
[VMS]
/httpd/-/admin/*  #localhost,~daniel,r+w
```

7. 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
```

9.3.1 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 five departments, each of which wishes to have two representatives capable of administering the departmental Web information. There is one authentication database, named after the site:

```
HYPOTHETICAL
```

In that database are twelve names, the first two the overall Web administrators, capable of administering all authorized paths, another two from each department makes up another ten:

```
WEB1        r+w
WEB2        r+w
JOHN        r+w
PAUL        r+w
GEORGE      r+w
RINGO       r+w
CURLY       r+w
LARRY       r+w
MOE         r+w
SHEMP       r+w
MAC         r+w
MYER        r+w
```

The authorization configuration file then contains:

```
[HYPOTHETICAL]
# allow web masters (!) to use the administration menu
#                       to revise web configuration files
# world has no access (read or write)
# access is only allowed from a browser on the same system as the HTTPd
/httpd/-/admin/*  #localhost,web1,web2,r+w
/ht_root/local/*  #localhost,web1,web2,r+w
# allows BOGUS_R_AND_D department representatives to maintain their web
# this may only be done from within the company subnet
# world has read access
/web/dept/bogus_R_and_D/*   150.15.30.*,John,Paul,web1,web2,r+w ; r
# and so on for the rest of the departments
/web/dept/real_R_and_D/*    150.15.30.*,George,Ringo,web1,web2,r+w ; r
# (the next uses line continuation just for illustration)
/web/dept/marketing/*       150.15.30.*,\
                            Curly,Larry,web1,web2,r+w ;\
                            read
/web/dept/production/*      150.15.30.*,Moe,Shemp,web1,web2,r+w ; r
/web/dept/inventory/*       150.15.30.*,Mac,Myer,web1,web2,r+w ; r
[WORLD]
# we need a spot for general POSTing (world has no read access)
# (just for the purposes of illustration :^)
/web/world/*  r+w
```

9.4 Databases, Authentication and Group

At this point it should be stressed that the HTTPd-specific (HTA) authentication and group databases are identical in internal structure and administration, it is merely the role they assume that differentiates them. The role of the database is purely a function of where they are used in the configuration file realm directive, the first mentioned becomes the authentication database, any subsequent, second database assumes the role of group database (otherwise that role is also assumed by the authentication database).

Authentication Database

An authentication, or realm database contains usernames and associated passwords. Passwords are verified using this database. **Passwords are always stored in encrypted form.**

When using an HTTPd-specific database it can also be used to contain per-user permissions, that control access to configured paths. In this role it acts as both realm (authentication) and group (authorization) database.

It is also possible for the system's SYSUAF database to be used for authentication purposes. Of course HTTPd-specific permissions cannot be stored in this database, but if using SYSUAF-authenticated usernames a VMS security profile can be used to control access to served files, see Section 9.6.

Warning ... when using Digest authentication, an authentication database cannot simply be renamed to represent a group-name change, as the database name is encrypted as part of the MD5 Digest!

Authorization Database

An authorization, or group database, contains a collection of usernames and their associated permissions within the group. That is, can that user read and/or write paths configured against that group. A group database is never used for password verification (authentication).

An authorization database is not always necessary. If only a small number of users need to be authorized against a given path they can be specified individually using a ''~username'' format (see Access Restriction List).

Database Administration

The same server administration menu is used to administer authentication/authorization databases, see Section 9.5 and Section 11.3.

There is a thirty-one character limit on database names, and a fifteen character limit on usernames and passwords. All database and user names are case-insensitive. *BASIC* authentication scheme passwords are case insensitive, however *DIGEST* authentication scheme passwords should all be either upper or lower case, mixed case will not work.

9.5 Server Authorization Administration

The same server administration menu is used to maintain both the authentication and authorization databases, see Section 11.3.

The server authorization databases can be initially set-up by **breaking-in to the server**, see Section 11.1

A server's currently loaded path authorization and user authentication information may be administered and interrogated on-line using the server administration facility. See Chapter 11 for further detail.

9.6 SYSUAF-Authenticated Users

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

Authentication Identifiers

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=ID 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!

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.

   ```
   map /http/-/change/* /http/-/change/*
   ```

   The authorization configuration file must provide authenticated access.

   ```
   [VMS]
   /httpd/-/change/vms/* https:,r+w
   ```

   Also see Section 9.8.

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>
```

Note that it is possible to create nil-access VMS accounts that may be used solely for WASD server authentication. Create a standard VMS account in a group dedicated for this function, specifying /NOINTERACTIVE /NONETWORK /NOBATCH /FLAG=DISMAIL.

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 (see Section 11.4).

SYSUAF and SSL

When SSL is in use (see Chapter 10) 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 Authentication Policy.

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.

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. 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 (see Section 11.4).

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.

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.

9.7 Controlling Server Write Access

Write access by the server into VMS directories (using the POST or PUT HTTP methods) is controlled using VMS ACLs. **This is in addition to the path authorization of the server itself of course!** The requirement to have an ACE on the directory 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.

9.8 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.

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

```
map /http/-/change/* /http/-/change/*
```

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

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

[ANOTHER_GROUP]
/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 email, for the change to be made on your behalf.
</I></BLOCKQUOTE>

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

Chapter 10
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 10.4, for further information on SSL technology.

Enhancing privacy with SSL, certificate management and varying browser behaviours is not for the faint-hearted!

The *Secure Sockets Layer* protocol (SSL) is designed to provide privacy between two communicating applications, in the case of HTTP between the browser (client) and the HTTPd (server). It provides other capabilities, such as client authentication, but these are not considered here. SSL operates by establishing an authenticated, encrypted communication path between the two applications, ''wrapping'' the entire application protocol, such as HTTP, in a 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.

WASD implements SSL using a freely available software library known as ''SSLeay'', see Section 10.3.

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 patent issues and USA export restrictions on some cryptographic technology that would impact the basic package's potential for download out of the USA. For further comment on this aspect check the SSLeay FAQ section ''Is this legal?''. No recommendations are made by this author on the use of the **SSLeay package** in any particular locality, except to comment that it **is in commercial and non-commercial use in many applications across Europe, North America and Australasia**.

The WASD SSL component is available as a separate, easily integrated package, either with complete SSLeay source or just SSLeay object libraries, some SSLeay utility object modules for building executables and WASD support files. Obtain these from the same source as the main package.

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.

SSL Access Control

When authorization is in place (see Chapter 9) access to username/password controlled data/functionality benefits enormously from the privacy of an authorization environment inherently secured via the encrypted communications of SSL. SSL may be used as part of the site's access control policy, as whole-of-site, see Authentication Policy, or on a per-path basis, see Conditionals and Access Restriction List.

Interoperability

WASD SSL has been tested against the following browsers (if any others are tested please advise the result).

Browser	Version	OK	Comment
Netscape Navigator	v3.0 (Digital Unix)	yes	accepts CA (both ‘‘.PEM’’ and ‘‘.CRT’’), initiates server certificate dialog on first ‘‘https:’’ connect
Netscape Navigator	v3.02 (WinNT)	yes	accepts CA (both ‘‘.PEM’’ and ‘‘.CRT’’), initiates server certificate dialog on first ‘‘https:’’ connect
Netscape Navigator	v3.03 (VMS)	yes	accepts CA(both ‘‘.PEM’’ and ‘‘.CRT’’), initiates server certificate dialog on first ‘‘https:’’ connect
Netscape Navigator	v4.03 (WinNT)	yes	accepts CA (both ‘‘.PEM’’ and ‘‘.CRT’’), initiates server certificate dialog on first ‘‘https:’’ connect

Browser	Version	OK	Comment
Netscape Navigator	v4.05 (Digital Unix)	yes	accepts CA (both ‘‘.PEM’’ and ‘‘.CRT’’), initiates server certificate dialog on first ‘‘https:’’ connect
MS Internet Explorer	3.02 (WinNT)	no	accepts CA (‘‘.CRT’’ only), but then refuses to transact SSL requests claiming the site is not certified by a well-known CA!
MS Internet Explorer	4.0 (WinNT)	yes	accepts CA (‘‘.CRT’’ only), initiates server certificate dialog on first ‘‘https:’’ connect
Cryptozilla	08-APR-1998 (WinNT)	yes	This in-development browser, implements full-strength encryption (USA domestic grade), and demonstrates WASD SSL’s full-strength capability. In Cryptozilla’s case the DES-CBC3-SHA cipher, using a 168-bit key, was negotiated.

10.1 SSL Configuration

SSL installation creates the major directory HT_ROOT:[SRC.SSLEAY-0_9_0b] containing at the least the SSLeay copyright notice, object libraries, object modules for building executables, example certificates, and some other support files and documentation. If a complete source package has been obtained the tree is considerably larger.

The example HTTPd startup procedure already contains support for the SSL executable. If this has been used as the basis for startup then only the respective boolean needs to be altered to use the SSL 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.

The following example illustrates creating two standard HTTP services, one on the default port 80, another on port 8000, and two SSL services, the first on the default port 443, the second on port 8443.

```
[Service]
alpha.wasd.dsto.defence.gov.au
alpha.wasd.dsto.defence.gov.au:8000
https://alpha.wasd.dsto.defence.gov.au
https://alpha.wasd.dsto.defence.gov.au:8443
```

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.

The one further requirement of an SSL server is a *certificate*. This is located using the HTTPD$SSL_CERT logical name during startup.

SSL Quick-Start

If the basic WASD v5.*n* package is installed, configured and performing satisfactorily the following steps give a quick outline for support of SSL.

1. UNZIP the WASD SSL package. 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
   ```

2. Once UNZIPped it will be necessary to link the executables. This should be performed using the UPDATE.COM

   It assumes a *vanilla* setup, using the standard directories and account environment described in this document. All sections prompt before performing any action and default to ''no''. The SSL parameter indicates the SSL components should be processed.

   ```
   $ SET DEFAULT HT_ROOT:[000000]
   $ @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 FREEWARE_DEMO.COM procedure. It should detect the TCP/IP agent in use (if not specify UCX or NETLIB as a parameter). It is necessary to specify that it is to use the SSL executable. Follow the displayed instructions.

   ```
   $ @HT_ROOT:[000000]FREEWARE_DEMO SSL
   ```

4. Modify server startup procedures.

   - **Using pre-existing, non-v5.*n* Startup**

     1. Modify the server startup procedure to INSTALL and the HTTPD80.COM procedure (or equivalent) to use the HTTPD_SSL.EXE executable (or HTTPD_SSL_NETLIB.EXE as appropriate)

     2. Modify the server startup procedure to define a logical HTTPD$SSL_CERT locating the example certificate at HT_ROOT:[EXAMPLE]HTTPD.PEM, for example

        ```
        DEFINE /SYS HTTPD$SSL_CERT HT_ROOT:[EXAMPLE]HTTPD.PEM
        ```

   - **Using v5.*n* Startup**

     Modify the ''PROVIDE_SSL'' symbol to be 1 (or true).

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]
   alpha.dsto.defence.gov.au
   https://alpha.dsto.defence.gov.au
   ```

6. Shutdown the server completely, then restart.

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

   - **Standard HTTP**

     ```
     http://host.domain/
     ```

   - **SSL HTTP**

     ```
     https://host.domain/
     ```

8. Once the server has been proved functional with the example certificate it is recommended that a server-specific certificate be created using the CA.COM procedure described in Section 10.2. This may then be used by placing it in the appropriate local directory, changing the HTTPD$SSL_CERT logical definition in the startup and restarting the server.

10.2 Certificate Management

This section is not a tutorial on certificates and their management. Refer to the listed references, Section 10.4, for further information on this aspect.

Certificates associate a public cryptographic key with the identity of the certificate holder, usually an individual or server. It includes a distinguishing 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 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. This is the default used by the example startup procedure and will suffice to demonstrate the server's functionality. 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 HT_ROOT:[SRC.SSLEAY-0_9_0b.WASD]CA.COM procedure, which allows the user to act as a certificate authority (CA) and issue identifying certificates. Because the CA is non-authoritative (not a well-known CA) these certificates have little value except to allow SSL transactions to be established with trusting clients.

This procedure allows CA as well as server certificates to be issued, as well as providing *foreign command* symbols for other certificate management utilities. For usage requirements and parameters check the commentary in the procedure preamble.

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*) and MS Internet Explorer (v4.*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
```

The following would load the WASD testing CA certificate in a format suitable for

- Navigator (v3.*n*, v4.*n*):

  ```
  /ht_root/src/ssleay-0_9_0b/wasd/cacert.pem
  ```

- MS Internet Explorer (v4.*n*):

  ```
  /ht_root/src/ssleay-0_9_0b/wasd/cacert.crt
  ```

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* seems able to use the ‘‘.CRT’’ certificate.

Changing Server or CA 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.

10.3 SSLeay

WASD implements SSL using a freely available software library known as ‘‘SSLeay’’ (pronounced ‘‘S-S-L-E-A-Y’’, i.e. all letters spelt), authored and copyright by Eric Young and Tim Hudson. It is not a GNU-licensed package, but does makes unrestricted commercial and non-commercial use available. The FAQ for SSLeay may be found at

http://www.psy.uq.oz.au/~ftp/Crypto/

This should be consulted for all information on the SSL technology employed by WASD.

If the SSLeay component of WASD is installed it can be found at

HT_ROOT:[SRC.SSLEAY-0_9_0b]

The previous version of SSLeay, 0.8.1, provided with WASD v5.0.0, can also be linked and will work with WASD v5.1.0. It is preferable to move to SSLeay v0.9.0b as known bugs in the previous version have been fixed in this one (ignoring the issue of new bugs being introduced ;^). If v0.8.1 is installed it will be found at

HT_ROOT:[SRC.SSLEAY-0_8_1]

It has been necessary to make minor modifications to the 0.9.0b distribution to support VMS (there was rudimentary support that looked like a hang-over from a previous distribution). These changes are very minor and designed to address the differences in VMS and DECC versions. All changes to source can be found by searching for ‘‘MGD’’ in [...]*.C, [...]*.H and [...]*.COM files.

These changes have been made only to support WASD's use of the package, they are not proposed as general SSLeay modifications, **i.e. they were purely pragmatic!**

Our thanks to the authors of and contributors to the SSLeay package, although if there is a documentation boffin out there please contact eric@CryptSoft.com :^)

10.4 SSL References

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

- http://www.netscape.com/newsref/std/SSL.html
  Netscape Communication Corporation's SSL Overview
- http://www.psy.uq.oz.au/~ftp/Crypto/
  SSLeay FAQ.
- http://www.camb.opengroup.org/www/prism/wwwj/index.html
  A good introduction to SSLeay.
- http://www.camb.opengroup.org/~fjh/Papers/cook/ssl_cook.html
  An excellent overview of managing certification with SSLeay.
- http://www.consensus.com/security/ssl-talk-faq.html
  SSL-Talk discussion list FAQ

Chapter 11
Server Administration

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 this can also be administered on-line through a browser.

A menu of server administration functions is provided by the server.

online hypertext link

In addition a Web **Upd**ate facility allows some administration of file system portions of the Web. See Chapter 15.

If authorization is in use (see Chapter 9) it is recommended that 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 (start the server using the /SYSUAF qualifier):

```
[VMS]
/httpd/-/admin/*  #localhost,~username,r+w
```

If SSL is in use (see Chapter 10) 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 administration menu it is necessary to comply with all the requirements for Web update of files. This is discussed in general terms in Chapter 15. 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 administration menu and reports while restricting the ability to intiate 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
```

11.1 Breaking-in To The Server!

The server’s authentication/authorization environment can be circumvented under special circumstances. **Note that this is not possible without administrator action and persists only as long as the administrator actually performs the action!**

This provision exists for only four foreseeable situations:

1. ad hoc server configuration
2. initial configuration of the authentication environment
3. loss of the administrator’s password
4. demonstration/experimentation purposes

Method

If performing the initial authentication configuration

make sure the HT_AUTH logical is correctly defined .

Manually start a new instance of the server on a non-standard port using the /PROMISCUOUS qualifier. This can be done at the command line. There is now a security hole. Example:

```
$ HTTPD = "$HT_EXE:HTTPD.EXE"
$ SPAWN /WAIT HTTPD /NOLOG /SERVICE=8088 /PROMISCUOUS
```

This server with then allow access using any username/password combination. Even if not completely paranoid it’s perhaps a good idea to append a specific password to the qualifier, the server will then only authenticate to that.

```
$ SPAWN /WAIT HTTPD /NOLOG /SERVICE=8088 /PROMISCUOUS=VERYCAUTIOUS
```

Access this instance of the server with a browser and use the server administration menu.

```
http://host:port/httpd/-/admin/
```

It is now possible to review server-generated reports, check rule mappings, create databases, enter username details, change the administrator’s password to something known, etc., etc. Shutdown that server again (ctrl-Y) and the security hole disappears.

Note that if a site authorization rule already maps the administration menu path or configuration file path the server may report duplicate path errors. This is due to the /PROMISCUOUS startup effectively introducing the following two rules, ensuring the menu and configuration files can be accessed regardless of anything already in or missing from the authorization file.

```
[PROMISCUOUS]
# access to the server administration menu
/httpd/-/admin/* r+w
# access to write into the "usual" location for configuration files
/ht_root/local/* r+w
```

Other rules may be placed in the local configuration file for the [PROMISCUOUS] realm specifically for use during this mode of access. They will not apply during normal server operation. Be sure they are placed before other rules in the file.

11.2 HTTPd Server Reports

The server provides a number of internally generated reports.

It is possible to use this facility standalone, without configuring authorization, see Section 11.1.

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.
2. **Configuration -** A tabular summary of the server's current configuration. This is a convenient method for viewing the information from the HTTPD$CONFIG file.
3. **Message -** A tabular report of the server's current message database, multiple languages shown if configured that way.
4. **Mapping -** All loaded mapping rules.

   In addition a query-form interface allows the user to input a path and watch the rules as the server resolves it. Note that conditionals are not applied to rules when doing this checking (generally not possible such a situation), and so these cannot be assessed. As a result it is possible to see multiple paths indicated as matched. The resulting VMS specification and/or script specification will indicate the actual mapping.
5. **Path Authorization -** If authorization is in use (see Chapter 9) this report lists the paths with associated authorization and access control.
6. **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.
7. **Cache -** Allows monitoring of cache behaviour and performance, as well as the files currently in the cache. For further information see Chapter 13.
8. **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!

9. **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.

10. **Memory -** Provides a report and does an integrity check on each of the Virtual Memory (VM) zones employed by the WASD HTTPd.

11. **Request -** Lists current requests (always shows at least your own connection accessing this report :^) and if enabled by configuration a history list of the most recent requests. Enabled by the configuration parameter [RequestHistory].

12. **SSL -** The SSL report lists counts of the number of SSL transactions initiated and completed, along with session cache statistics. If the current request was via SSL it also lists the ciphers available and current session information.

13. **Activity -** Provide a graphical *snapshot* of server activity of a given period.

    This is dynamic data, held in memory only, and so is current only from the latest server startup. It is enabled by the configuration parameter [ActivityDays]. The administration menu 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 during the period by a line graph. A *greyed* area indicates no data available for that time (i.e. before the latest server startup, or in the future).

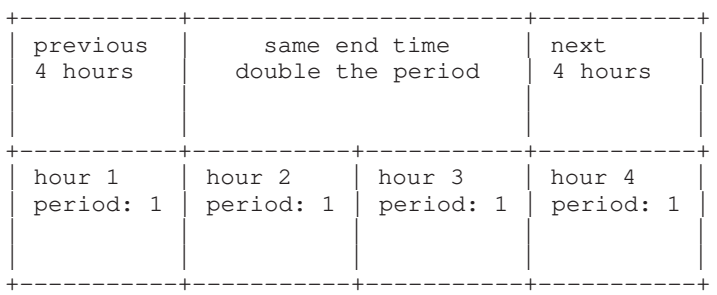
    Activity data is accumulated on a per-minute basis. This is the maximum granularity of any report. When reports are selected that can display less than this one minute granularity (i.e. with periods greater than four hours) the value shown is the **peak** of the number of minutes sampled for display. This better represents the load on the server than would a mean of those samples.

    For browsers recognised as capable (with v4.3 of the server these are Netscape Navigator 3.0ff and Microsoft Internet Explorer 3.02ff) this report is JavaScript-enabled, and if in focus the browser refreshes itself at an interval appropriate to the reporting period. If not in focus, the report is automatically refreshed when the browser is brought back into focus.

    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.

    The following example shows the layout for a four hour report. 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.



11.3 HTTPd Server Revise

The server provides a limited configuration revision facility.

1. **Configuration -** A form-driven interface allows the current configuration of the server to be altered on-line. 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.
2. **Message -** No form-driven interface is currently available for changing the messages. However it is possible to directly edit and update the on-disk file. The server can then be restarted to use the modified database (see Section 11.4).
3. **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 (see Section 11.4).
4. **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 (see Section 11.4).
5. **User Authentication -** User authentication comprises a number of dialogues that allow the 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 9 covers authentication detail.

11.4 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 5.3.

Control Section

1. **Server Restart/restartNOW/Exit/exitNOW - Caution ... no confirmation!** 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.
2. **Logging On/Off/Flush -** The HTTPD$LOG logical must be configured to allow access logging to be enabled and disabled from this menu.
3. **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.

Configuration *Action* Section

1. **Statistics Zeroed -** All counters are zeroed (except the *number-of-times-zeroed* counter!)
2. **Mapping Rules Reload -** Reloads the path mapping rules from the on-disk file into the running server.

   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.
3. **Path Authorization Reload -** Reloads the path authorization directives from the on-disk file into the running server.
4. **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 12
Scripting

This chapter is **not a tutorial on authoring CGI scripts**. There exists a plague of references in the popular computing press covering aspects of this technology, usually quite comprehensive. This chapter merely outlines the WASD implementation details, which are in general very much vanilla CGI.

Scripts are mechanisms for creating simple HTTP services, sending data to (and sometimes receiving data from) a client, extending the capabilities of the basic HTTPd. Anything that can write to SYS$OUTPUT can be used to generate script output. A DCL procedure or an executable can be the basis for a script. Simply TYPE-ing a file can be provide script output. Scripts execute in processes separate from the actual HTTP server but under it's control and interacting with it.

NOTE: WASD can manage a script's process environment either as a subprocess spawned by the HTTP server, or as a network process created using DECnet. By default it supports subprocess-based CGI scripts without further configuration. If DECnet-based CGI scripting or OSU (DECthreads) emulated scripting is desired see Section 12.8.

Scripts are enabled using the *exec* or *script* rules in the mapping file (see Chapter 8). The script portion of the *result* must be a URL equivalent of the physical VMS procedure or executable specification.

12.1 Caution!

Scripts are executed within unprivileged subprocesses spawned by the HTTP server. These subprocesses are owned by the HTTP server account (HTTP$SERVER). Script actions can potentially affect server behaviour. For example it is possible for a script to issue an ''HTTPD/DO=EXIT=NOW'' command, or to create or modify logical name values in the JOB table (e.g. change the value of LNM$FILE_DEV altering the logical search path). Obviously these types of actions are undesirable. In addition scripts can access any WORLD-readable and modify any WORLD-writable resource in the system/cluster, opening a window for information leakage or mischievous/malicious actions (some might argue that anyone with important WORLD-accessable resources on their system deserves all that happens to them

- but we know they're out there :^) Script authors should be aware of any potential side-effects of their scripts and Web administrators vigilant against possible malicious behaviours of scripts they do not author.

As of version 4.2 it has become possible to exercise some control over the privileges of spawned subprocesses, allowing enviroments that require scripts to have minimum privileges (e.g. NETMBX, TMPMBX for IPC) to provide them using the server account's authorized privileges. See Chapter 6.

12.2 Scripting Environment

WASD HTTPd scripting underwent a major redesign between v4.1 and v4.2. This was to provide a faster and more efficient scripting environment. It provided the opportunity for a much needed review of the DCL mechanism within the server. As a result two capabilities not found in earlier versions became available, persistant subprocesses (see below) and CGIplus (see Section 12.7).

Process creation under the VMS operating system is notoriously slow and expensive. This is an inescapable overhead when scripting via child processes. An obvious strategy is to avoid, at least as much as possible, the creation of subprocesses. The only way to do this is to share subprocesses between multiple scripts/requests, addressing the attendant complications of isolating potential interactions between requests. These could occur through changes made by any script to the subprocess' enviroment. For VMS this involves symbol and logical name creation, and files opened at the DCL level. In reality few scripts need to make logical name changes and symbols are easily removed between uses. DCL-opened files are a little more problematic, but again, in reality most scripts doing file manipulation will be images.

A reasonable assumption is that for almost all environments scripts can quite safely share subprocesses with great benefit to response latency and system impact (see Section 14.2 for a table with some comparative performances). If the local environment requires absolute script isolation for some reason then this subprocess-persistance may easily be disabled with a consequent trade-off on performance.

NOTE: With the form of subprocess management used in v4.2 and following, BYTLM can become an issue. When setting the HTTPd account BYTLM quota allow approxiamtely 12,500 bytes per subprocess that can be concurrently active, plus a general allowance (technically, allow 1.0 x /NETBUF= plus 1.0 x + 0.5 x + 0.5 x /SUBBUF=). That is if the subprocess *hard-limit* (see below) is 20 then BYTLM should be set to at least 250,000 plus 50,000. Of course in such a case PRCLM should be set to at least 20, preferably 40. These and other relevant quotas may be monitored using the HTTPDMON utility or the server administration menu.

Zombies

The term *zombie* is used to describe subprocesses when persisting between uses (the reason should be obvious, they are neither ''alive'' (processing a request) nor are they ''dead'' (deleted) :^) Zombie subprocesses have a finite time to exist (*non*-life-time?) before they are automatically purged from the system (see Chapter 6). This keeps process clutter on the system to a minimum.

12.3 Script Run-Time

Scripts are merely executed or interpreted files. Although by default VMS executables and DCL procedures can be used as scripts, other run-time environments may also be configured. For example, scripts written for the Perl language may be transparently given to the Perl interpreter in a script subprocess. This type of script activation is based on a unique file type (extension following the file name), for the Perl example this is most commonly ''.PL'', or sometimes ''.CGI''. Both of these may be configured to automatically invoke the site's Perl interpreter, or any other run-time environment for that matter.

This configuration is performed using the [DclScriptRunTime] parameter, where a file type is associated with a run-time environment. This parameter takes two components, the file extension and the run-time verb. The verb may be specified as a simple, globally-accessable verb (e.g. one embedded in the CLI tables), or in the format to construct a *foreign-verb*, providing reasonable versatility. Run-time parameters may also be appended to the verb if desired. The server ensures the verb is foreign-assigned if necessary, then used on a command line with the script file name as the final parameter to it.

The following is an example showing a Perl run-time environment being specified. The first line assumes the ''Perl'' verb is globally accessable on the system (e.g. perhaps provided by the DCL$PATH logical) while the second (for the sake of illustration) shows the same Perl interpreter being configured for a different file type using the foreign verb syntax.

```
[DclScriptRunTime]
.PL PERL
.CGI $PERL_EXE:PERL
```

A file contain a Perl script then may be activated merely by specifying a path such as the following

```
/cgi-bin/example.pl
```

To add any required parameters just append them to the verb specified.

```
[DclScriptRunTime]
.XYZ XYZ_INTERPRETER -vms -verbose -etc
.XYZ $XYZ_EXE:XYZ_INTERPRETER /vms /verbose /etc
```

If a more complex run-time environment is required it may be necessary to *wrap* the script's execution in a DCL procedure.

Script File Extensions

The WASD server does not require a file type (extension) to be explicitly provided when activating a script. This can help hide the implementation detail of any script. If the script path does not contain a file type the server searches the script location for a file with one of the known file types, first ''.COM'' for a DCL procedure, then ''.EXE'' for an executable, then any file types specified using script run-time configuration directive, in the order specified.

For instance, the script activated in the Perl example above could have been specified as below and (provided there was no ''EXAMPLE.COM'' or ''EXAMPLE.EXE'' in the search) the same script would have been executed.

```
/cgi-bin/example
```

12.4 CGI Compliance

The HTTPd scripting mechanism is designed to be WWW CGI (Common Gateway Interface) compliant, based in part on by the INTERNET-DRAFT authored by D.Robinson (drtr@ast.cam.ac.uk), 8 January 1996.

CGI Compliant Variables

Environment variables are created in a similar way to the CERN VMS HTTPd implementation, where CGI environment variables are provided to the script via DCL global symbols. Each CGI variable symbol name is prefixed with ''WWW_'' (by default, although this can be changed using the ''/CGI_PREFIX'' qualifier, see Section 5.3, this is not recommended if the WASD VMS scripts are to be used, as they expect CGI variable symbols to be prefixed in this manner).

Extensions to CGI Variables

In line with other CGI implemenations, additional, non-compliant variables are provided to ease CGI interfacing. These provide the various components of the query string. A *keyword* query string and a *form* query string are parsed into separated variables, named

```
WWW_KEY_number
WWW_KEY_COUNT
WWW_FORM_form-element-name
```

See the example below.

CGI Variable Capacity

DCL symbol values are limited to approximately 1024 characters. The CGI interface will provide symbols with values up to that limit if required. This should be sufficient for most circumstances.

CGI Variable Descriptions

Remember, all variables are prefixed by ''WWW_''.

Name	Description		''Standard'' CGI
AUTH_GROUP	authentication group (or empty)	no	
AUTH_REALM	authentication realm (or empty)	no	
AUTH_TYPE	authentication type (BASIC or DIGEST)		yes
CONTENT_LENGTH	''Content-Length:'' from request header		yes

Name	Description	''Standard'' CGI
CONTENT_TYPE	''Content-Type:'' from request header	yes
FORM_field	query string ''&'' separated form elements	no
GATEWAY_INTERFACE	''CGI/1.1''	yes
HTTP_ACCEPT	any list of browser-accepted content types	optional
HTTP_ACCEPT_CHARSET	any list of browser-accepted character sets	optional
HTTP_ACCEPT_LANGUAGE	any list of browser-accepted languages	optional
HTTP_AUTHORIZATION	any from request header	optional
HTTP_COOKIE	any cookie sent by the client	optional
HTTP_FORWARDED	any proxy/gateway hosts that forwarded the request	optional
HTTP_HOST	host and port request was sent to	optional
HTTP_IF_MODIFIED_SINCE	any last modified GMT time string	optional
HTTP_PRAGMA	any pragma directive of request header	optional
HTTP_REFERER	any source document URL for this request	optional
HTTP_USER_AGENT	client/browser identification string	optional
KEY_n	query string ''+'' separated elements	no
KEY_COUNT	number of ''+'' separated elements	no
PATH_INFO	virtual path of data requested in URL	yes
PATH_TRANSLATED	VMS file path of data requested in URL	yes
QUERY_STRING	un-URL-decoded string following ''?'' in URL	yes
REMOTE_ADDR	IP host address of HTTP client	yes

Name	Description	‘‘Standard’’ CGI
REMOTE_HOST	IP host name of HTTP client	yes
REMOTE_USER	authenticated remote user name (or empty)	yes
REQUEST_METHOD	‘‘GET’’, ‘‘PUT’’, etc.	yes
REQUEST_SCHEME	‘‘http:’’ or ‘‘https:’’	no
REQUEST_TIME_GMT	GMT time request received	no
REQUEST_TIME_LOCAL	Local time request received	no
SCRIPT_NAME	name of script being executed (e.g. ‘‘/query’’)	yes
SERVER_GMT	offset from GMT (e.g. ‘‘+09:30’’)	no
SERVER_NAME	IP host name of server system	yes
SERVER_PROTOCOL	HTTP protocol version (always ‘‘HTTP/1.0’’)	yes
SERVER_PORT	IP port request was received on	yes
SERVER_SOFTWARE	software ID of HTTP server	yes

CGI Variable Demonstration

The basic CGI symbol names are demonstrated here with a call to a script that simply executes the following DCL code:

```
$ SHOW SYMBOL WWW_*
$ SHOW SYMBOL *
```

Note how the request components are represented for ‘‘ISINDEX’’-style searching (third item) and a forms-based query (fourth item).

online hypertext demonstration

CGI Compliant Output

Script output must behave in a CGI-compliant fashion (by way of contrast, see Section 12.5). That is, a CGI script may redirect the location of the document, using a *Location:* header line, or may supply a data stream beginning with a *Content-Type:* header line. Both must be followed by a blank line.

If the script output begins with a CGI-compliant ‘‘Content-Type: text/ . . . ’’ (text document) the HTTPd assumes that output will be line-oriented and requiring HTTP carriage-control (each record/line terminated by a line-feed), and will thereafter ensure each record it receives is correctly terminated before passing it to the client. In this way DCL procedure output (and the VMS CLI in general) is supported transparently. Any other content-type is assumed to be *binary* and no carriage control is enforced.

12.4.1 Example DCL Scripts

A simple script to provide the system time might be:

```
$ say = "write sys$output"
$! the next two lines make it CGI-compliant
$ say "Content-Type: text/plain"
$ say ""
$! start of plain-text script output
$ show time
```

A script to provide the system time more elaborately (using HTML):

```
$ say = "write sys$output"
$! the next two lines make it CGI-compliant
$ say "Content-Type: text/html"
$ say ""
$! start of HTML script output
$ say "<HTML>"
$ say "Hello ''WWW_REMOTE_HOST'"  !(CGI variable)
$ say "<P>"
$ say "System time on node ''f$getsyi("nodename")' is:"
$ say "<H1>''f$cvtime()'</H1>"
$ say "</HTML>"
```

12.5 Raw HTTP Output

A script does not have to output a CGI-compliant data stream. If it begins with a HTTP header status line (e.g. ''HTTP/1.0 200 OK''), HTTPd assumes it will supply a **raw** HTTP data stream, containing all the HTTP requirements. This is the equivalent of the *no-parse-header*, or ''nph . . . '' named scripts of some environments.

Any such script must observe the HyperText Transfer Protocol. Every line must be terminated by a *carriage-return* and *line-feed* (represented as ''\r''\n''), or as a minimum by a single *line-feed*. In particular, the type of the data being returned by the scripts must be included in an HTTP header sent prior to the data itself. Headers for the two most common data types will be illustrated here. Note that the blank line is strictly necessary, it terminates the header.

Plain-Text

```
HTTP/1.0 200 ok\r\n
Content-Type: text/plain\r\n
\r\n
```

HTML

```
HTTP/1.0 200 ok\r\n
Content-Type: text/html\r\n
\r\n
```

Raw HTTP DCL Script

The following example show a non-CGI-compliant DCL script similar in function to the CGI-compliant one above. Note the full HTTP header and each line explicitly terminated with a carriage-return and line-feed pair.

```
$ cr[0,8] = %x0d
$ lf[0,8] = %x0a
$ say = "write sys$output"
$! the next line determines it is raw HTTP stream
$ say "HTTP/1.0 200 Success''cr'''lf'"
$ say "Content-Type: text/html''cr'''lf'"
$! response header separating blank line
$ say "''cr'''lf'"
$! start of HTML script output
$ say "<HTML>''lf'"
$ say "Hello ''WWW_REMOTE_HOST'''lf'"
$ say "<P>''lf'"
$ say "Local time is ''WWW_TIME_LOCAL'''lf'"
$ say "</HTML>''lf'"
```

Raw HTTP C Script

When scripting using the C programming language and providing a full HTTP response there can be considerable efficiencies to be gained by providing a binary output stream from the script. This may be simply provided using a code construct similar to following to reopen <stdout> in binary mode.

```
/* reopen output stream so that the '\r' and '\n' are not filtered */
#ifdef __DECC
   if ((stdout = freopen ("SYS$OUTPUT", "w", stdout, "ctx=bin")) == NULL)
      exit (vaxc$errno);
#endif
```

This is used consistently in WASD scripts. Of course after that the full HTTP header must be supplied.

```
   fprintf (stdout,
"HTTP/1.0 200 Success\r\n\
Content-Type: text/html\r\n\
\r\n\
<HTML>\n\
Hello %s\n\
<P>\n\
System time is %s\n\
</HTML>\n",
   getenv("WWW_REMOTE_HOST"),
   getenv("WWW_TIME_LOCAL"));
```

12.6 Raw HTTP Input

The logical name SYS$INPUT (with a synonym of HTTP$INPUT for backward compatibility), <stdin> for C Language based scripts, defines a mailbox providing a stream containing the request body (if any). This is available for procedures and executables to explicitly open and read.

Note that this is a raw stream, and HTTP *lines* (carriage- return/line-feed terminated sequences of characters) may have be blocked together for network transport. These would need to be explicity parsed by the program.

NOTE: Versions of the server prior to 4.3 supplied the full request (header then body) to the script. This was not fully CGI-compliant. Versions 4.3 and following supply only the body, although the previous behaviour may be explicitly selected using the configuration parameter [DclFullRequest].

12.7 CGIplus Scripting

Common Gateway Interface ... plus lower latency, plus greater efficiency, plus far less system impact!

I know, I know! The term CGIplus is a bit too cute but I had to call it something!

CGIplus attempts to eliminate the overhead associated with creating the subprocess and then executing the image of a CGI script. It does this by allowing the subprocess and any associated image/application to continue executing between uses, eliminating any startup overheads. This reduces both the load on the system and the request latency. In this sense these advantages parallel those offered by commercial HTTP server-integration APIs, such as Netscape NSAPI and Microsoft ISAPI, without the disadvantages of such proprietory interfaces, the API complexity, language dependency and server process integration.

CGIplus is not as complex (and consequently nor as versatile) as another approach to improving CGI performance, Open Market's *FastCGI*, see http://www.fastcgi.com/

CGIplus design is generic enough to be easily implemented by other server architectures if found desirable. (For example, it is imagined Unix platforms would implement the CGIplus variable stream using *named pipes* one of which would be designated by the CGIPLUSIN environment variable.) The CGIplus-specific script environment and example code has been made as platform-neutral as possible, providing potential for a more wide-spread adoption. Existing CGI scripts can rapidly and elegantly be modified to additionally support CGIplus. The capability of scripts to easily differentiate between and operate in both standard CGI and CGIplus environments with a minimum of code revision offers great versatility.

CGIplus Performance

A simple performance evaluation indicates the advantage of CGIplus. See Section 14.2 for some test results comparing the non-persistant-process, persistant-process and CGIplus environments.

Without a doubt, the subjective difference in activating the same script within the standard CGI and CGIplus environments is quite startling!

CGIplus Programming

The script interface is still CGI, which means a new API does not need to be learned and existing CGI scripts are simple to modify.

See examples in HT_ROOT:[SRC.CGIPLUS]

on-line hypertext link

Instead of having the CGI variables available from the environment (generally accessed via the C Language *getenv( )* standard library call) a CGIplus script must read the CGI variables from CGIPLUSIN. They are supplied as a series of records (lines) containing a CGI variable name (in upper-case), an equate symbol and then the variable value. The line will never contain more than 1024 characters. The format may be easily parsed and as the value contains no encoded characters may be directly used.

Requirements when using:

- The read will block between subsequent requests and so may be used to coordinate the application.
- The first record read in any request can always be discarded. This is provided so that a script may be synchronized outside of the general CGIplus variable read loop (the DCL and Perl examples use this feature).
- The CGIplus variable stream should be completely read (up until the blank line, see below) before beginning any request processing.
- The CGIplus variable stream should ALWAYS BE COMPLETELY READ (up until the blank line, see below).
- An empty record (blank line) indicates the end of a single request's CGIplus variable stream. Reading MUST be halted at this stage. Request processing may then commence.

After processing, the CGIplus script can loop, waiting to read the details of the next request from CGIPLUSIN.

Request output (to the client) is written to SYS$OUTPUT (<stdout>) as per normal CGI behaviour. **End of output MUST be indicated by writing a special EOF record to the output stream.** This is bit of a *kludge*, and the least elegant part of CGIplus design, but it is also the simplest implementation. A unique EOF sequence is generated for each use of DCL via a zombie or CGIplus subprocess. A non-repeating series of bits most unlikely to occur in normal output is employed . . . but there is still a very, very, very small chance of premature termination of output (one in 2^280 I think!) See HT_ROOT:[SRC.HTTPD]DCL.C for how the value is generated.

The CGIplus EOF string is obtained by the script from the logical name CGIPLUSEOF, defined in the script subprocess' process table, using the scripting language's equivalent of F$TRNLNM( ), SYS$TRNLNM( ), or a getenv( ) call (in the C standard library). This string will always contain less than 64 characters and comprise only printable characters. It must be written at the conclusion of a request's output to the output stream as a single record (line) but may also contain a <CR><LF> or just <LF> trailing carriage-control (to allow for programming language requirements). It only has to be evaluated once, as the processing begins, remaining the same for all requests over the life-time of that instance of the script.

HTTP input (raw request stream, header and any body) is still available to a CGIplus script.

Code Examples

Of course a CGIplus script should only have a single exit point and should explicitly close files, free allocated memory, etc., after processing a request (i.e. not rely on image run-down to clean-up after itself). It is particularly important when modifying existing scripts to work in the CGIplus environment to ensure this requirement is met (who of us hasn't thought ''well, this file will close when the image exits anyway''?)

It is a simple task to design a script to modify it's behaviour according to the environment it is executing in. Detecting the presence or absence of the CGIPLUSEOF logical is sufficient indication. The following C code fragment shows simultaneously determining whether it is a standard or CGIplus environment (and setting an appropriate boolean), and getting the CGIplus EOF sequence (if it exists).

```
int  IsCgiPlus;
char  *CgiPlusEofPtr;

IsCgiPlus = ((CgiPlusEofPtr = getenv("CGIPLUSEOF")) != NULL);
```

The following C code fragment shows a basic CGIplus request loop, reading lines from CGIPLUSIN, and some basic processing to select required CGI variables for request processing.

```
if (IsCgiPlus)
{
   char  *cptr;
   char  Line [1024],
         RemoteHost [128];
   FILE  *CgiPlusIn;

   if ((CgiPlusIn = fopen (getenv("CGIPLUSIN"), "r")) == NULL)
   {
      perror ("CGIplus: fopen");
      exit (0);
   }

   for (;;)
   {
      /* will block waiting for subsequent requests */
      for (;;)
      {
         /* should never have a problem reading CGIPLUSIN, but */
         if (fgets (Line, sizeof(Line), CgiPlusIn) == NULL)
         {
            perror ("CGIplus: fgets");
            exit (0);
         }
         /* first empty line signals the end of CGIplus variables */
         if (Line[0] == '\n') break;
         /* remove the trailing newline */
         if ((cptr = strchr(Line, '\n')) != NULL) *cptr = '\0';

         /* process the CGI variable(s) we are interested in */
         if (!strncmp (Line, "WWW_REMOTE_HOST=", 16))
            strcpy (RemoteHost, Line+16);
      }

      (process request, signal end-of-output)
   }
}
```

CGI scripts can write output in record (line-by-line) or binary mode (more efficient because of buffering by the C RTL). When in binary mode the output stream must be flushed immediately before and after writing the CGIplus EOF sequence (note that in binary a full HTTP stream must also be used). This code fragment shows placing a script output stream into binary mode and the flushing steps.

```
/* reopen output stream so that the '\r' and '\n' are not filtered */
if ((stdout = freopen ("SYS$OUTPUT", "w", stdout, "ctx=bin")) == NULL)
   exit (vaxc$errno);

do {

   (read request ...)

   /* HTTP response header */
   fprintf (stdout, "HTTP/1.0 200 ok\r\nContent-Type: text/html\r\n\r\n");

   (other output ...)

   if (IsCgiPlus)
   {
      /* the CGIplus EOF must be an independant I/O record */
      fflush (stdout);
      fprintf (stdout, "%s", CgiPlusEofPtr);
      fflush (stdout);
   }

} while (IsCgiPlus);
```

If the script output is not binary (using default <stdout>) it is only necessary to ensure the EOF string has a record-delimiting new-line.

```
fprintf (stdout, "%s\n", CgiPlusEofPtr);
```

Other languages may not have this same requirement. DCL procedures are quite capable of being used as CGIplus scripts.

See examples in HT_ROOT:[SRC.CGIPLUS]

on-line hypertext link

Whenever developing CGIplus scripts/applications (unlike standard CGI) don't forget that after compiling, the old image must be purged from the server before trying out the new!!! (I've been caught a number of times :^)

Scripting subprocesses may be purged or deleted using (see Section 5.3.2.4):

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

Other Considerations

Multiple CGIplus scripts may be executing in subprocesses at any one time. This includes multiple instances of any particular script. It is the server's task to track these, distributing appropriate requests to idle subprocesses, monitoring those currently processing requests, creating new instances if and when necessary, and deleting the least-used, idle CGIplus subprocesses when configurable thresholds are reached. Of course it is the script's job to maintain coherency if multiple instances may result in resource conflicts or race conditions, etc., between the scripts.

The CGIplus subprocess can be given a finite life-time set by configuration parameter (see Chapter 6). If this life-time is not set then the CGIplus will persist indefinitely (i.e. until purged due to soft-limits being reached, or explicitly purged/deleted). When a life-time has been set the CGIplus subprocess is automatically deleted after being idle for the specified period (i.e. not having processed a request). This can be useful in preventing sporadically used scripts from cluttering up the system indefinitely.

In addition, an idle CGIplus script can be terminated by the server at any time the subprocess soft-limit is reached (the subprocess SYS$DELPRC( )ed) so resources should be largely quiescent when not actually processing. Of course a CGIplus subprocesses may also be manually terminated from the command line (e.g. STOP/ID=).

Some CGIplus scripting information and management is available via the server administration menu, see Section 11.2.

CGIplus Rule Mapping

CGIplus scripts are differentiated from standard CGI scripts in the mapping rule configuration file using the ‘‘script+’’ and ‘‘exec+’’ directives. See Chapter 8.

Scripts capable of operating in both standard CGI and CGIplus environments may simply be accessed in either via rules such as

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

while specific scripts can be individually designated as CGIplus using

```
script+ /cgiplus_example* /cgi-bin/cgiplus_example*
```

Caution! If changing CGIplus script mapping it is advised to restart the server rather than reloading the rules. Some conflict is possible when using new rules while existing CGIplus scripts are executing.

12.8 DECnet Scripting

‘‘Imitation is the sincerest form of flattery’’ - proverb

NOTE: WASD requires no additional configuration to support subprocess-based scripting. The following information applies only if DECnet-based scripting is desired.

By default WASD executes scripts within subprocesses, but can also provide scripting using DECnet for the process management. DECnet scripting is not provided to generally supplant the subprocess-based scripting but augment it for certain circumstances:

- To provide an environment within WASD where OSU-based scripts (both CGI and OSU-specific) may be employed without modification.
- To allow nodes without a full HTTP service to participate in providing resources via a well-known server, possibly resources that only they have access to.
- Load-sharing amongst cluster members for *high-impact* scripts or particularly busy sites.
- Provide user-account scripting.

DECnet Performance

Any DECnet based processing incurs some overheads.

- connection establishment
- NETSERVER image activation
- NETSERVER maintenance (such as logs, etc.)
- activation of DECnet object image or procedure
- DECnet object processing
- activation by object of image or procedure
- DECnet object run-down
- NETSERVER image reactivation on completion of object processing

As of version 5.2 WASD provides reuse of DECnet connections for both CGI and OSU scripting, in-line with OSU v3.3 which provided reuse for OSU scripts. This means multiple script requests can be made for the cost of a single DECnet connection establishment and task object activation. This functionality provides substantial performance improvements as indicated by Section 14.3. Note that the OSU task procedure requires the definition of the logical name WWW_SCRIPT_MAX_REUSE representing the number of times a script may be reused. The WASD startup procedures can provide this.

In practice both the WASD CGI and OSU scripts seem to provide acceptable responsiveness.

Rule Mapping

DECnet-based scripts are mapped using the same rules as subprocess-based scripts, using the SCRIPT and EXEC rules (see Chapter 8 for general information on mapping rules). DECnet scripts have a DECnet node and *task specification string* as part of the mapping rule. There are minor variations within these to further identify it as a WASD or an OSU script. See Section 12.8.4 for information on mapping user scripting.

The specification string follows basic VMS file system syntax (RMS), preceding the file components of the specification. The following example illustrates declaring that paths beginning with FRODO will allow the execution of scripts from the CGI-BIN:[000000] directory on DECnet node FRODO.

```
exec /FRODO/* /FRODO::/cgi-bin/*
```

In similar fashion the following example illustrates a script ‘‘frodo_show’’ that might do a ‘‘SHOW SYSTEM’’ on node FRODO. Note that these rules are case-insensitive.

```
script /frodo-showsys /frodo::/cgi-bin/showsys.com
```

Both of the above examples would use the WASD CGI DECnet environment (the default if no task specification string is provided). By including task information other environments, in particular the OSU scripting enviroment, can be specified for the script to be executed within. The default task is named CGIWASD and can also be explicitly specified (although this behaviour would be the same as that in the first example)

```
exec /frodo/* /frodo::"task=cgiwasd"/cgi-bin/*
```

All task specification strings may also use zero as the task abreviation.

```
exec /frodo/* /frodo::"0=cgiwasd"/cgi-bin/*
```

To execute a script within the OSU environment specify the standard OSU task executive WWWEXEC, as in the following example:

```
exec /osu/* /FRODO::"task=wwwexec"/cgi-bin/*
```

This would allow any URL beginning with ‘‘/osu/’’ to execute a script in the OSU environment.

Local System

To specify any script to execute on the same system as the HTTP server specify the node name as zero or SYS$NODE.

```
exec /decnet/* /0::"task=cgiwasd"/cgi-bin/*
exec /osu/* /sys$node::"task=wwwexec"/cgi-bin/*
```

Mapping rules are included in the examples (see HT_ROOT:[EXAMPLE]) providing this. After the DECnet environment has been started any CGI script may be executed on the local system via DECnet by substituting ‘‘/decnet/’’ for ‘‘/cgi-bin/’’ as the script path, and any OSU script available by using ‘‘/osu/’’. Behaviour is indeterminate, though it shouldn’t be catastrophic, if one is invoked using the incorrect path (i.e. an OSU script using /decnet/ or a CGI script using /osu/).

12.8.1 Script System Environment

The target system must have sufficient of the WASD server environment to support the required CGI script activation and activity. If the target system is actually the same system as the HTTP server then it already exists, or if part of the local system’s cluster, then providing this should be relatively straight-forward. If the target system has none of the server environment then at a minimum it must have the logical name CGI-BIN defined representing the directory containing the required DECnet object procedure and scripts. The following fragment illustrates this:

```
$ DEFINE /SYSTEM /TRANSLATION=(CONCEALED) CGI-BIN device:[dir.]
```

In this directory must be located the WASDCGI.COM and WWWEXEC.COM procedures required by the network task. Of course other parts of the environment may need to be provided depending on script requirements.

12.8.1.1 Proxy Access

The local system must have *proxy access* to each target scripting system (even if that ‘‘target’’ system is the same system as the HTTP server). This involves creating a proxy entry in each target hosts’s authorization database. The following example assumes the existance of a local HTTP$SERVER account. If it does not exist on the target node then one must be created with the same security profile as the HTTP server’s.

Caution! If unsure of the security implications of this action consult the relevant VMS system management security documentation.

The zero represents the system the server is currently executing on.

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD /PROXY 0::HTTP$SERVER HTTP$SERVER /DEFAULT
```

It is necessary to ensure the account has permission to write into it's home directory. A network process creates a NETSERVER.LOG (Phase-IV) or NET$SERVER.LOG (DECnet-Plus) file in the home directory, and will fail to start if it cannot!

12.8.1.2 DECnet Objects

To provide DECnet scripting DECnet *object(s)* must be specified for any system on which the scripts will be executed. The DECnet object is the program or procedure that is activated at the target system inside of a network-mode process to interact with the HTTP server.

DECnet-Plus (OSI/Phase-V)

DECnet-Plus uses the NCL utility to administer the network environment. The following NCL scripting shows the creation of a network application for the WASD CGI object:

```
$ MCR NCL
CREATE NODE 0 SESSION CONTROL APPLICATION CGIWASD
SET NODE 0 SESSION CONTROL APPLICATION CGIWASD ADDRESSES = {NAME=CGIWASD} -
,CLIENT =  -
,INCOMING ALIAS = TRUE -
,INCOMING PROXY = TRUE -
,OUTGOING ALIAS = FALSE -
,OUTGOING PROXY = TRUE -
,NODE SYNONYM = TRUE -
,IMAGE NAME = CGI-BIN:[000000]CGIWASD.COM -
,INCOMING OSI TSEL =
```

To create a DECnet-Plus OSU WWWEXEC object:

```
$ MCR NCL
SET NODE 0 SESSION CONTROL APPLICATION WWWEXEC ADDRESSES = {NAME=WWWEXEC} -
,CLIENT =  -
,INCOMING ALIAS = TRUE -
,INCOMING PROXY = TRUE -
,OUTGOING ALIAS = FALSE -
,OUTGOING PROXY = TRUE -
,NODE SYNONYM = TRUE -
,IMAGE NAME = CGI-BIN:[000000]WWWEXEC.COM -
,INCOMING OSI TSEL =
```

These must be executed at each system (or server) startup, and may be executed standalone, as illustrated, or incorporated in the NCL script SYS$STARTUP:NET$APPLICATION_STARTUP.NCL for automatic creation at each system startup. Examples may be found in HT_ROOT:[EXAMPLE].

Phase-IV

DECnet Phase-IV uses the NCP utility to administer the network environment. The following NCP commands may be used each time during server startup to create the required DECnet objects. With Phase-IV the SET verb may be replaced with a DEFINE verb and the commands issued just once to permanently create the objects (a SET must also be done that first time to create working instances of the DEFINEd objects).

To create a DECnet CGI object:

```
$ MCR NCP
SET OBJECT CGIWASD NUMBER 0 FILE CGI-BIN:[000000]CGIWASD.COM
```

To create a DECnet OSU WWWEXEC object:

```
$ MCR NCP
SET OBJECT WWWEXEC NUMBER 0 FILE CGI-BIN:[000000]WWWEXEC.COM
```

Examples may be found in HT_ROOT:[EXAMPLE].

12.8.1.3 Reducing Script Latency

Script system network process persistance may be configured using NETSERVER logical names. These can control the number and quiescent period of the server processes. These logical names must be defined in the LOGIN.COM of the HTTP server account on the target script system.

- **NETSERVER$SERVERS_*username* -** This logical controls the number of network server processes that are kept available at any one time. Defining this logical results in a minimum of the specified number of quiescent server processes maintained. This can improve script response latency by circumventing the need to create a process to service the request, at the cost of cluttering the system with NETSERVER processes.

  ```
  DEFINE /JOB NETSERVER$SERVERS_HTTP$SERVER 5
  ```

- **NETSERVER$TIMEOUT -** This logical controls the duration a quiescent network process persists before being deleted. The default period is five minutes. The following examples first show reducing that to thirty seconds, the second increasing it to one hour. Again, this can improve script response latency by circumventing the need to create a process to service the request, at least during the period a previously created process continues to exist.

  ```
  DEFINE /JOB NETSERVER$TIMEOUT "0 00:00:30"
  DEFINE /JOB NETSERVER$TIMEOUT "0 01:00:00"
  ```

12.8.1.4 DECnet/OSU Startup

The example STARTUP.COM and STARTUP_DECNET.COM procedures found in the HT_ROOT:[EXAMPLE] directory provide the essentials for DECnet/OSU scripting. If the INSTALL.COM startup environment is used setting the PROVIDE_DECNET symbol to 1 in STARTUP.COM will create the DECnet scripting environment during server startup.

12.8.2 CGI

CGI scripts may be transparently executed within the DECnet scripting environment. This means that the script is executed within a network process, on the target system (which could be the local system), instead of within a subprocess on the local system. Other than that the WASD DECnet CGI environment behaves identically to the standard subprocess CGI environment. CGIplus scripting is not supported and if CGIplus-only scripts are executed the behaviour is indeterminate.

An example of making the HELP database on a system other than that hosting the HTTP server (using the CONAN script) would be done using the mapping rules

```
map /FRODO/help /FRODO/help/
script /FRODO/help/* /FRODO::/cgi-bin/conan/*
```

and for the example DCL SHOW script

```
script /FRODO/show* /FRODO::/cgi-bin/show*
```

12.8.3 OSU (DECthreads) Emulation

The OSU, or *DECthreads*, server is the most widely deployed VMS HTTP server environment, authored by David Jones and copyright the Ohio State University. See http://kcgl1.eng.ohio-state.edu/www/doc/serverinfo.html for more information.

The WASD HTTP server provides an emulation of the OSU scripting environment. This is provided so that OSU-based scripts (both CGI-style and OSU-specific) may be employed by WASD with no modification. As this emulation has been designed through examining OSU code and lots of trial and error it's behaviour may be incomplete or present errors. A list of OSU scripts known to work with WASD is provided at the end of this section, see Known Working Scripts.

Supported scripts include only those that depend on the OSU WWWEXEC object and dialog for all functionality. Any script that uses other OSU-specific functionality is not supported. Interactions between WASD's and OSU's authentication/authorization schemes may be expected.

Please remember this is a first-cut of reverse-engineered technology. The author would like to know of any OSU scripts the WASD emulation *barfs* on, and will attempt to address the associated limitation(s) and/or problem(s).

OSU Setup

Software necessary for supporting the OSU scripting environment (e.g. WWWEXEC.COM) and selected OSU scripts (mainly for testing purposes) have been extracted from the OSU v3.3a package and included in the HT_ROOT:[SRC.OSU] directory. This has been done within the express OSU licensing conditions.

```
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.
```

An example DECnet and OSU scripting startup may be found in HT_ROOT:[EXAMPLE]. This should be called from or used within the HTTP server startup. It includes two logical definitions required for common OSU scripts. Other tailoring may be required for specific OSU scripts.

OSU - General Comments

David Jones, the author of the DECthreads (OSU) HTTP server, outlines his reasons for basing OSUs scripting on DECnet (reproduced from a USENET NEWS reply to a comment this author made about DECnet-based scripting).

```
------------------------------------------------------------------------

From            JONESD@er6.eng.ohio-state.edu (David Jones)
Organization    The Ohio State University
Date            12 Aug 1997 09:04:11 GMT
Newsgroups      vmsnet.sysmgt,comp.os.vms,comp.infosystems.www.servers.misc
Message-ID      <5sp8ub$brs$1@charm.magnus.acs.ohio-state.edu>

------------------------------------------------------------------------

 . . .   some text omitted

Since I was comfortable with DECnet, I based the scripting system
for the OSU server around it.    The key reasons to use netserver
processes rather than spawning sub-processes:

    1. DECnet automatically caches and re-uses netserver processes,
       whereas there were well-known performance problems with spawning
       sub-processes.

    2. DECnet processes are detached processes, so you don't worry about
       the effect of scripts consuming pooled quotas (e.g. bytlm) on
       the HTTP server process.

    3. Creation/connection with the DECnet server process is asynchronous
       with respect to the server so other operations can proceed concurrently.
       Spawning is done in supervisor mode, blocking the server's operation
       until the child process is completely initialized.

    4. With DECnet, scripts can be configured to run on different nodes
       for load balancing.

    5. In addition to the standard 'WWWEXEC' object, you can create
       other 'persistent' DECnet objects that the server communicates with
       as scripts. (this was implemented years before OpenMarket's FastCGI
       proposal).

    6. CGI is not the be-all end-all of scripting.  The dialog phase of
       OSU's scripting environment allows scripts to do things CGI
       is incapable of, such as ask the server to translate an arbitrary
       path and not just what followed the script name in the URL.

People grouse all the time about the installation difficulties caused by
it's reliance on DECnet,  the reason shown above were cited to show that it
wasn't made so capricously.

 . . .   some text omitted
```

```
David L. Jones                  |       Phone:    (614) 292-6929
Ohio State Unviversity          |       Internet:
2070 Neil Ave. Rm. 122          |                jonesd@kcgl1.eng.ohio-state.edu
Columbus, OH 43210              |                vman+@osu.edu

Disclaimer: Dogs can't tell it's not bacon.
```

The OSU server's DECnet scripting is not based on arbitrary considerations. This author does not disagree with any of the concerns, and as may be seen from WASD documentation the design of WASD also directly addresses points 1, 3 and 5 with the use of persistant subprocesses and CGIplus. Certainly DECnet-based scripting addresses the very legitimate point 4 (and also allows nodes with specific resources to participate without installing full HTTP server environments). For all practical purposes point 2 may be addressed by adjusting process quotas. Point 6 is only too true (possibly at least until Java servers and *servlets* become ubiquitous :^)

Known Working Scripts

The following is a list of OSU-specific scripts that the WASD v5.1 implementation has either been developed or tested against, and any installation notes or other WASD specifics. The author would like to know of any OSU scripts the WASD emulation has problems or works successfully with.

- All of the scripts, etc. provided in the HT_ROOT:[SRC.OSU] directory. These include:
  - cgi_symbols
  - cgi-mailto
  - html_preproc
  - set_dcl_env
  - testcgi
  - testform
  - tmail
  - vmshelpgate
  - webbook
- **helpgate**

  Comment out the *Conan The Librarian* mappings for the ''/help'' path and provide the following in HTTPD$MAP:

  ```
  # first make "/help" into a script specification
  map /help* /htbin/helpgate/help*
  # general rule mapping "/htbin" to OSU DECnet scripts
  exec /htbin/* /0::"0=wwwexec"/cgi-bin/*
  # map the non-script part of the path back to just "/help"
  pass /htbin/helpgate/help* /help*
  ```

  It is possible to support both HELP environments (although *helpgate* will not work without owning the ''/help'' path), merely provide another mapping for *Conan* with a slightly different path, for example:

```
map /chelp /chelp/
script /chelp/* /cgi-bin/conan/*
```

- **HTML pre-processor**

  Yes, backward compatibility can be provided for those old OSU .HTMLX files in your new WASD environment ;^) All that is needed is a file type mapping to the script in the HTTPD$CONFIG configuration file.

  ```
  [AddType]
  .HTMLX  text/html  /htbin/html_preproc  OSU SSI HTML
  ```

- showtime
- mgmt

12.8.4 User Scripts

The WASD DECnet environment provides a simple mechanism for executing scripts within accounts other than the server's. This allows configured users to write and maintain scripts within their own areas and have them execute as themselves. Both standard CGI and OSU scripting may be provided for with this facility.

Of course there is always a down-side. Be careful to whom this capability is granted. User scripts are executed within a user network-mode process created by DECnet. Script actions cannot generally affect server behaviour, but they can access any WORLD-readable and modify any WORLD-writable resource in the system/cluster, opening a window for information leakage or mischievous/malicious actions. Script authors should be aware of any potential side-effects of their scripts and Web administrators vigilant against possible destructive behaviours of scripts they do not author.

User scripting is not enabled by default. To provide this facility mapping rules into the user area must be provided in much the same way as for user directories, See Section 8.6.

The ''EXEC'' rule provides a wildcard representation of users' script paths. As part of this mapping a subdirectory specifically for the web scripts should always be included. **Never** map users' top-level directories. For instance if a user's account home directory was located in the area WWW_USER:[DANIEL] the following rule would potentially allow the user DANIEL to provide scripts from the home subdirectory [.WWW.CGI-BIN] using the accompanying rules (first for CGI, second for OSU scripts):

```
exec /~*/cgi-bin/* /0"":/www_user/*/www/cgi-bin/*
exec /~*/osu-bin/* /0"":"0=wwwexec"/www_user/*/www/cgi-bin/*
```

Scripts located in these directories are accessable via paths such as the following:

```
/~daniel/cgi-bin/test
```

Explicit User Account

Using mapping rules it is possible to explicitly specify the user account for a particular script or scripts to be executed within. This may be useful if an application has quota or other resource requirements that are desired to be withheld from the HTTP server account (i.e. it can provide a measure of isolation between the server and application accounts).

```
exec /whatever-bin/* /0"WHATEVER"::/whatever_root/cgi-bin/*
script /dowhatever/* /0"WHATEVER"::/whatever_root/cgi-bin/dowhatever/*
```

Proxy Access

For each user account permitted to execute local scripts proxy access to that account must be granted to the HTTP server account.

Caution! 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 /PROXY <node>::HTTP$SERVER <account>
```

For example, the following would allow the HTTP server to execute scripts on behalf of the username DANIEL.

```
UAF> ADD /PROXY 0::HTTP$SERVER DANIEL
```

12.9 Java Scripts

Java classes may be used to perform CGI scripting with WASD. They may be designed as standard CGI scripts (with the inevitable latency of the class loading) or as CGIplus scripts (with the attendant benefit of lower latency).

Note that Java scripts must always be mapped and executed using the CGIplus path, however some can behave as standard CGI scripts, exiting after responding to the request, while others can persist, responding to multiple requests (see Section 12.7). The CGIplus path is always necessary as Java does not have direct access to a process' general environment, the traditional way of passing CGI variables, so the WASD implementation uses the CGIplus data stream to provide CGI information.

WASD provides a class to allow a relatively simple interface to the CGI environment for both GET and POST method scripts. This and a collection of demonstration scripts may be found in the HT_ROOT:[SRC.JAVA] directory.

Developed using the first-release JDK1.1 beta kit for OpenVMS Alpha V7.1.

Requirements

- Ensure the Java class file type is mapped to the Java run-time in the HTTPD$CONFIG configuration file.

```
[DclScriptRunTime]
.CLASS  @CGI-BIN:[000000]JAVA.COM
```

- The following content types are configured, also in HTTPD$CONFIG.

```
[AddType]
.CLASS  application/octet-stream  -  Java class
.JAVA  text/plain  -  Java source
.JAR  application/octet-stream  -  Java archive
.PROPERTIES  text/plain  -  Java properties
```

- The CGI-BIN logical includes the HT_ROOT:[JAVA] class directory in the server startup.

```
$ JAVA_ROOT = F$TRNLNM("HT_ROOT") - ".]" + ".JAVA.]"
$ DEFINE /SYSTEM /TRANSLATION=(CONCEALED) -
         CGI-BIN 'EXE_ROOT','SCRIPT_LOCAL_ROOT', -
         'SCRIPT_ROOT','JAVA_ROOT'
```

12.10 HTTP Persistant-State Cookies

The WASD server is *cookie-aware*. That is, if the client supplies a ''Cookie:'' request header line it is passed to a CGI script as ''WWW_HTTP_COOKIE'' CGI variable symbol. If a cookie is not part of the request this symbol does not exist. A script may use the ''Set-Cookie:'' response header line to set cookies.

Here is a small demonstration of cookie processing using a DCL script.

online hypertext demonstration

Chapter 13
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.

Note that it is a file-system cache. Only documents generated from the file system are cached, not from any potentially dynamic sources, such as scripts, directory listings, SSI documents, etc. The reason should be obvious. Only the file system provides a reliable mechansim for ensuring the validity of the cached data (i.e. has the original changed in some way since loaded?)

Files are cached according to **mapped** path (not necessarily the same path supplied with the request) and not by the file name represented by any path. This is a design decision targeted at avoiding any access to RMS before searching the cache. For example, the ambiguous reference to the directory

```
/ht_root/
```

may result in the following file being accessed (due to home page resolution)

```
HT_ROOT:[000000]HOME.HTML
```

and the contents returned to the client and consequently cached. Each time the path ''/ht_root/'' is subsequently requested it will be path hit and serviced from the cache entry without any recourse to RMS.

Of course the same file may be requested with the unambigious path

```
/ht_root/home.html
```

which is completely different to the first instance, although ultimately accessing the same file. Hence one file may be cached multiple times against distinct paths. Although isolated instances of this are possible, the likelihood of significant resources being consumed in practice should be quite low.

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 noticable. 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. The author's feeling is though, that for most VMS sites high levels of hits are not a great concern, and for these caching can easily be left disabled, particularly if the file system's virtual I/O cache is enabled.

A comparison between cached and non-cached performance is provided in Chapter 14.

Why take so long to implement caching? (introduced in version 4.5) Well, WASD's intranet services are not particularly busy sites. This coupled with powerful hardware has meant performance has not been an overriding concern. However, this cache module came about because I felt like creating it and it was an obvious lack of functionality within the server, not because WASD (the organisation) needed it.

Terminology

This is what is meant when used.

- ''hit'' refers to a request path being found in cache. If the data is still valid the request can be supplied from cache.
- ''load''ing the cache refers to reading the contents of a file into cache memory.
- ''valid'' means that the file from which the cached data was originally read has not had it's revision date changed (the implication being is the file contents have not changed.

13.1 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 administration menu's cache report.

Two sets of data provide complementary information, cache activity and file request profile.

- **Activity information** 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-cachable 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 information** 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.

Recommendation

Monitor the site's cache behaviour and adjust parameters from an assessment based on the guidelines above.

Again, the author's suggestion is, that for most VMS sites, high levels of access are not a great concern, and for these caching can easily be left disabled.

13.2 Cache Content Validation

The cache will automatically revalidate the 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.

Cache validation is also always performed if the request uses ''Pragma: no-cache'' (i.e. as with the Netscape Navigator *reload* 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.

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 either from the server administration menu or using command line server control, as in the following example

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

13.3 Cache Configuration

The cache is controlled using HTTPD$CONFIG file configuration directives. A number of parameters control the cache's behaviour. See the example configuration file HT_ROOT:[EXAMPLE]HTTPD$CONFIG.CONF.

- **Cache** enables and disables caching.
- **CacheChunkKBytes** sets the granularity of memory blocks allocated to file data (in kilobytes). To reduce the number of, and possible blocks sizes of memory, being requested (potentially improving memory allocation performance) data allocated to cache is done in multiples of this chunk size.
- **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 never be cached.

- **CacheFrequentHits** 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 ''CacheFrequentHits'' 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.
- **CacheHashTableEntries** sets the size of the hash table, used to rapidly index into the cached paths. Generally bigger is better (each entry consumes four bytes, hence 1024 entries consume 4096 bytes). Due to the hashing algorithm employed the size should be in magnitudes of two (e.g. 1024, 2048, etc.), if not, the specified amount is *rounded down* to the nearest (e.g. 1000 to 512).
- **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 re-validation is forced with a *reload*.

13.4 Cache Control

The cache may be enabled, disabled and purged from the server administration menu (see Chapter 11). In addition the same control may be exercised from the command line (see Section 5.3.2) 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.

Chapter 14
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 (ignoring scripts which execute within their own context). An obvious improvement would be to have multi-processor threading or a pool of server processes, one per CPU, servicing requests. The latter may be the approach of future refinements.

The server has been tested with up to 30 concurrent requests originating from 6 different systems and continues to provide an even distribution of data flow to each client (albeit more slowly :^)

Test results are all obtained using the native Digital TCP/IP Services executable. The NETLIB image may provide very slightly lower results due to the additional NETLIB layer. These results are indicative only!

Simple File Request Turn-Around

Two sets of data are now reported, one with caching disabled, the other enabled.

A series of tests using batches of 200 accesses were made and the results averaged. The first test returned an empty file measuring response and file access time, without any actual transfer. The second and third requested files of 16K and 64K characters respectively, testing performance with more realistic scenarios. All were done using one and then ten concurrent requests.

The test system was a lightly-loaded AlphaServer 2100, VMS v7.1 and DEC TCP/IP 4.2. No *Keep-Alive:* functionality was employed so each request required a complete TCP/IP connection and disposal, although the WWWRKOUT utility (see Section 16.6) was used on the same system as the HTTP server, eliminating actual network transport. DNS (name resolution) was disabled. The command lines are show below.

```
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/ht_root/exercise/0k.txt"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/ht_root/exercise/0k.txt"
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/ht_root/exercise/16k.txt"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/ht_root/exercise/16k.txt"
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/ht_root/exercise/64k.txt"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/ht_root/exercise/64k.txt"
```

The following results were derived using the v5.2 server.

Cache Disabled

Response	Concurrent	Duration (seconds)	Requests/Second
0K	1	2.11	95
0K	10	1.70	117
16K	1	3.04	66
16K	10	2.50	80
64K	1	6.71	30
64K	10	5.10	39

Cache Enabled

Response	Concurrent	Duration (seconds)	Requests/Second
0KB	1	0.95	210
0KB	10	0.82	244
16KB	1	1.86	107
16KB	10	1.60	125
64KB	1	4.74	43
64KB	10	4.25	47

Significantly, with both environments, **throughput actually improves** at ten concurrent requests (probably 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 220% improvement. This is a fairly crude analysis, but does give some indication of cache efficiencies.

Simple File Request Transfer Rate

The simple text file request under similar conditions indicates a potential transfer rate **well in excess of 1 Mbyte per second**. (Remember, both client and server are on the same system, so the data, although being transported by TCP/IP networking, is not actually ending up out on a physical network.) This serves to demonstrate that server architecture should not be the limiting factor in file throughput.

```
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=10 -
/PATH="/sys$common/sysexe/tnt$server.exe"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=10 -
/PATH="/sys$common/sysexe/tnt$server.exe"
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=10 -
/PATH="/sys$common/sysexe/cxx$compiler.exe"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=10 -
/PATH="/sys$common/sysexe/cxx$compiler.exe"
```

The following results were derived using the v5.2 server.

Transfer Rate

Response	Concurrent	Duration (seconds)	Mbytes/Second
2.4MB (4700 blocks)	1	6.4	3.8
2.4MB (4700 blocks)	10	5.7	4.2
6.9MB (13442 blocks)	1	40	1.7
6.9MB (13442 blocks)	10	17	4.0

Significantly, there were no dramatic drops in transfer rate between one and ten concurrent requests! In fact an increase in throughput!

14.1 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.

14.2 Subprocess-based Scripting

Persistant-subprocesses are probably the most efficient solution for child-process scripting under VMS. See Section 12.2. 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 three scripting environments available. Two results are provided here. Both were obtained using the WWWRKOUT utility (see Section 16.6) accessing the same CGI test utility script, HT_ROOT:[SRC.CGIPLUS]CGIPLUSTEST.C, which executes in both standard CGI and CGIplus environments. A series of 200 access were made and the results averaged. The first test returned only the HTTP header, evaluating raw request turn-around time. The second test requested a body of 16K characters, again testing performance with a more realistic scenario. No *Keep-Alive:* functionality was employed so each request required a complete TCP/IP connection and disposal, although the WWWRKOUT utility was used on the same system as the HTTP server, eliminating actual network transport. DNSLookup (host name resolution) was disabled. The test system was a lightly-loaded AlphaServer 2100, VMS v7.1 and DEC TCP/IP 4.2.) The command lines are show below:

```
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/cgi-bin/cgiplustest?0"
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/cgi-bin/cgiplustest?16"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/cgiplus-bin/cgiplustest?0"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/cgiplus-bin/cgiplustest?16"
```

The following results were derived using the v5.2 server.

CGI Scripting

Response	Concurrent	Duration (seconds)	Requests/Second
0KB	1	14.7	13.65
0KB	10	8.60	23.3
16KB	1	13.69	14.6
16KB	10	8.68	23.0

CGIplus Scripting

Response	Concurrent	Duration (seconds)	Requests/Second
0KB	1	4.20	47.6
0KB	10	4.31	46.4
16KB	1	4.32	46.3
16KB	10	4.39	45.6

Again significantly, **throughput actually improved** at ten concurrent requests (probably due to the latency of the serial TCP/IP connection/disconnection in one-by-one, compared to several happening concurrently).

Although these results are indicative only, they do show CGIplus to have a potential for improvement over standard CGI in the order of 200%, 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 RDBS access languages).

14.3 DECnet-based Scripting

Using the same environment as when testing subprocess-based CGI scripts (see above) this series of tests assesses the performance of the same script being executed using DECnet to manage the processes. DECnet Phase-IV was in use on a VMS v7.1 AlphaServer 2100.

```
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/decnet/cgiplustest?0"
$ WWWRKOUT /SIM=1 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/decnet/cgiplustest?200"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/decnet/cgiplustest?0"
$ WWWRKOUT /SIM=10 /NOBREAK /NOVARY /NOHEAD /NOOUT /COUNT=200 -
/PATH="/decnet/cgiplustest?200"
```

The following results were derived using the v5.2 server, which now provides for DECnet connection reuse.

DECnet CGI Script - No Connection Reuse

Response	Concurrent	Duration (seconds)	Requests/Second
0KB	1	32.4	6.2
0KB	10	50.0	4.0
16KB	1	25.8	7.8
16KB	10	22.8	8.8

DECnet CGI Script - With Connection Reuse

Response	Concurrent	Duration (seconds)	Requests/Second
0KB	1	17.8	11.5
0KB	10	11.9	16.8
16KB	1	18.4	10.9
16KB	10	12.2	16.4

Some Observations

This section comments on non-persistant scripts (i.e. those that must run-up and run-down with each request - general CGI behaviour). As may be seen from comparing the two tables connection reuse offers distinct benefits in reduced response times, consistency of response times and overall thoughput, 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 (11.5 against 14.7 for single requests, 16.8 against 23.3 for ten concurrent requests).

While earlier versions cautioned on the use of DECnet-based scripting this has been relaxed somewhat through connection reuse.

14.4 SSL

At this time there are no definitive measurements of SSL performance (see Chapter 10), as work on an SSL version of the WWWRKOUT utility has not yet been undertaken. 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.

14.5 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]). This can slow processing significantly. Most log analysis tools can convert numeric addresses so DNS resolution is often an unnecessary burden.

   This can actually make a remarkable difference. The same test provided very different throughputs with DNS lookup enabled and disabled (v4.5 server, cache enabled).

	duration (seconds)	**requests/second**
DNSLookup ON	6.30	32
DNSLookup OFF	0.95	210

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 [StreamLFpaths]).
3. Use persistant-subprocess DCL/scripting (configuration parameter [ZombieLifeTime])
4. Use CGIplus-capable scripts whenever possible.

5. Enable caching (configuration parameter [Cache]).
6. Disable logging (configuration parameter [Logging]).
7. Set the HTTP server process priority higher, say to 6 (use startup qualifier /PRIORITY=).
8. Reduce to as few as possible the number of mapping rules.
9. Use a pre-defined log format (e.g. ‘‘common’’, configuration parameter [LogFormat]).
10. Disable request history (configuration parameter [RequestHistory]).
11. Disable activity statistics (configuration parameter [ActivityDays]).

Chapter 15
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

Update Access Permission

If SSL is in use (see Chapter 10) 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 9.7 for information on write access control for the server account.

Chapter 16
Utilities

Foreign commands for external utilities (and the HTTPD control functionality) will need to be assigned in the adminstration users' LOGIN.COM. Alternatives should be used when the environment supported through NETLIB. For example:

```
HTTPD == "$HT_EXE:HTTPD"
!HTTPD == "$HT_EXE:HTTPD_NETLIB"
HTTPDMON == "$HT_EXE:HTTPDMON"
!HTTPDMON == "$HT_EXE:HTTPDMON_NETLIB"
STREAMLF == "$HT_EXE:STREAMLF"
WWWRKOUT == "$HT_EXE:WWWRKOUT"
!WWWRKOUT == "$HT_EXE:WWWRKOUT_NETLIB"
```

16.1 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/
```

16.2 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/
```

16.3 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/
```

16.4 StreamLF Utility

This utility converts VARIABLE format files to STREAM_LF. 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. Use ‘‘STREAMLF/HELP’’ for some basic usage information.

NOTE This server can also be configured to automatically convert any VARIABLE record format files it encounters to STREAM_LF.

16.5 HTTPd Monitor

The HTTP server may be monitored using the HTTPDMON utility. This utility continuously displays a screen of information comprising three sections:

1. **Process Information**
   HTTPd process information includes its up-time, CPU-time consumed (excluding any subprocesses), I/O counts, and memory utilization.
2. **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. **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).

The following shows example output (after overnight server testing):

```
Port: 80    HTTPDMON v1.6.0 AXP (HTTPd v5.1)   Saturday,  4-JUL-1998 09:35:30

Process: HTTPd:80  PID: 20200467  User: HTTP$SERVER
     Up: 0 13:52:33.62  CPU: 0 06:21:11.35  Restart: 0
Pg.Flts: 2192  Pg.Used: 6%  WsSize: 9280 (4640kB)  WsPeak: 5888 (2944kB)
    AST:   13/600   BIO:    4/512   BYT:      0/122016  DIO:    2/512
    ENQ:    2/600   FIL:    4/100   PRC:      3/50       TQ:    6/200
```

```
Connect: 365756  SSL: 0 (0%)  Acc/Rej/Bsy: 365756 / 0 / 0  Cur/Peak: 0 / 15
  Parse: 378338  Error: 0  Forbidden: 0  Redirect-Rem/Loc: 0 / 12582
 DELETE: 0  GET: 341316  HEAD: 37022  POST: 0  PUT: 0
  Admin: 0  Cache: 8 / 113449 (0)  DECnet-CGI/OSU: 0 - 0 / 0
    DCL: 87408  CGI/plus: 50475 / 50437 (50202)  Spawn/Cur: 1767 / 3  Dir: 25237
   File: 12591 (0)  IsMap: 0  Menu: 0  Put: 0  SSI: 63093  Upd: 37856

    1xx: 0  2xx: 328403  3xx: 0  4xx: 25200  5xx: 12152  (0 errors)
     Rx: 30,002,610  Tx: 2,950,147,996  (bytes)

   Time: 04 08:35:15  Status: 200  Rx: 80  Tx: 24822  Duration: 2.003
Service: http://gamma.dsto.defence.gov.au:80
   Host: gamma.dsto.defence.gov.au (131.185.250.202)
Request: GET /tree/ht_root/*.*
```

The ‘‘/HELP’’ qualifier provides a brief usage summary.

This information is, in part, obtained from the following logical names:

- **HTTPD*port*$COUNT1**
- **HTTPD*port*$COUNT2**
- **HTTPD*port*$PID**
- **HTTPD*port*$REQUEST**

The server counter values are carried over when a server (re)starts (provided the system has stayed up). To reset the counters use the on-line server administration menu (see Chapter 11).

If [DNSlookup] is disabled for the HTTP server the HTTPDMON utility attempts to resolve the numeric address into a host name. This may be disabled using the /NORESOLVE qualifier.

16.6 Server Workout *(stress-test)*

The WWWRKOUT (‘‘World Wide Web Workout’’) utility exercises an HTTP server, both in the number of concurrent connections maintained and in the number of back-to-back sequential connection requests and data transfers. A native UCX image and a separate NETLIB image are provided.

This utility can be used to *stress-test* the WASD VMS HTTP server (or any other), or to make comparisons between it and other servers. When stress-testing a server, evaluating performance or just using it to try and break a test-bed server, it is best used from multiple, autonomous systems concurrent.

It sets up and maintains a specified number of concurrent connections to a server. It reads a buffer of data from each connection in turn, where data is waiting (does not block), until the document transfer is complete and the connection closed by the server. It then closes the local end and immediately reuses the now-free socket to initiate another sequence. If enabled (it is by default), the utility attempts to reflect the real-world in varying the data transfer rate for each connection, by setting the number of bytes read during each read loop differently for each connection. All transfered data is discarded.

The data transfer rate for each connection is displayed at connection close. It is by default it is the effective transfer rate, that is the rate from opening the connection to closing it, and so includes request processing time, etc. If the ''/NOEFFECTIVE'' qualifier is employed it measures the document data transfer rate only.

Although a single document path may be specified on the command line it is preferable to supply a range of document paths, one per line in a plain text file. Each document path and/or type specified should be different to the others, to exercise the server and file system cache. Any number of paths may be specified in the file. If the file is exhausted before the specified number of connections have been established the file contents are recycled from the first path. If a path or a file of paths is not specified the utility just continually requests the welcome document.

To assess a server's total throughput choose paths that lead to large documents (> 50K), where the overhead of connection setup, rule processing and transfer initiation are less significant than the data transfer itself. The buffer size variation functionality should be disabled using the ''/NOVARY'' qualifier when assessing data transfer rates. Responsiveness is better assessed using small documents (< 2K), where the overhead of the non-data-transfer activities is more significant.

WWWRKOUT [server_host_name[:port]] [path] [qualifiers]

- **/[NO]BREAK** will randomly break a small number of connections during the data transfer (tests server recovery under those circumstances)
- **/BUFFER=** number of bytes to be read from server each time (default is 1024, will be modified by the default ''/VARY'' qualifier)
- **/COUNT=** total number of connections and requests to be done (default is 100)
- **/[NO]EFFECTIVE** measures data transfer rate from request to close (if ''/NOEFFECTIVE'' is applied the rate is measured during data transfer only
- **/FILEOF=** file name containing paths to documents
- **/HELP** display brief usage information
- **/OUTPUT=** file name for output
- **/PATH=** single path to document to be retrieved
- **/PORT=** IP port number of HTTP server (default is 80)
- **/SERVER=** HTTP server host name
- **/SIMULTANEOUS=** number of simultaneous connections to be set up at any one time (default is 10)
- **/[NO]VARY** varies the size of the read buffer for each connection (default is vary)

Examples:

```
$ WWWRKOUT
$ WWWRKOUT www.server.host "/web/home.html"
$ WWWRKOUT www.server.host:8080 /FILEOF=PATHS.TXT
$ WWWRKOUT /SERVER=www.server.host /PORT=8080 /NOBREAK /NOVARY /FILEOF=PATHS.TXT
$ WWWRKOUT www.server.host:8080 /FILEOF=PATHS.TXT /NOEFFECTIVE /NOVARY
$ WWWRKOUT www.server.host /FILEOF=PATHS.TXT /COUNT=500 /SIMUL=20 /BUFFER=512
```

The ‘‘/HELP’’ qualifier provides a brief usage summary.