WASD VMS Web Services - Install and Config

November 2018

For version 11.3 release of WASD VMS Web Services.

Abstract

This document introduces the WASD Web Services package and provides detailed installation, update and configuration instructions.

For configuration and use of other significant WASD capabilities see ‘‘WASD Web Services - Features and Facilities’’

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

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

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

Author

Mark G. Daniel

Mark.Daniel@wasd.vsm.com.au

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

Online Search

online search

Online PDF

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

Online Demonstrations

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

WASD VMS Web Services

Copyright © 1996-2018 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 3 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.

WASD_ROOT:[000000]GNU_GENERAL_PUBLIC_LICENSE.TXT

http://www.gnu.org/licenses/gpl.txt

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

The Apache Group

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

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

Clark Cooper, et.al.

This package uses the Expat XML parsing toolkit.

```
Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
                               and Clark Cooper
Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Expat maintainers.

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

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

Bjoern Hoehrmann

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

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

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

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

Free Software Foundation

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

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

Ohio State University

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

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

OpenSSL Project

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

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

Paul E. Jones

This package uses SHA-1 hash code.

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

Freeware Public License (FPL)

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

RSA Data Security

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

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

Stuart Langridge

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

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

Tatsuhiro Tsujikawa

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

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

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

Contents

Chapter 1 New to WASD? Start Here!

Chapter 2 Installation and Update

Chapter 3 Server Account and Environment

Chapter 4 Configuration Considerations

Chapter 5 Security Considerations

Chapter 6 String Matching

Chapter 7 Conditional Configuration

Chapter 8 Global Configuration

Chapter 9 Service Configuration

Chapter 10 Message Configuration

Chapter 11 Cache Configuration

Chapter 12 Request Processing Configuration

Chapter 13 Authorization Configuration (Basics)

Chapter 1

New to WASD? Start Here!

Welcome!

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

1. **Unzip Package**

   Install the files following the guidelines in Chapter 2 **Note** that more than one archive may be needed (Source Archive, Object Module Archives).

   If **Transport Layer Security** (TLS - a.k.a. **Secure Sockets Layer** - SSL) is to be used and a TLS/SSL server image to be built the WASD [Open]SSL product must be installed at this stage (see "WASD Web Services - Features and Facilities"). An existing HP SSL1 (HP SSL is obsolete) for OpenVMS product requires no additional step. If the WASD [Open]SSL package it must UNZIPed into the [.WASD_ROOT] tree at this stage.

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

2. **INSTALL Package**

   Server installation is performed using the INSTALL.COM procedure (Section 2.7).

   - **Build Package -** Compile and link, or just link supplied object files to produce VMS executables for the system's version of VMS.
   - **Check Package -** Check basic operation of the package (Section 2.10).
   - **Create Server and Scripting Accounts -** Create two independent accounts, one for executing the server, the other for executing scripts (Section 3.1). If quotas are enabled on the target disk provides an ambit allocation for these accounts. Review this at some stage.
   - **Set Package Security -** This sections traverses the newly installed tree and sets all package directories and files to required levels of access (Section 5.3).
   - **Copy Support and Configuration Files -** Copy the example server support and configuration files (Section 3.3).

- **Install Scripts -** Selectively copy groups of scripts from package build directories into the scripting directories.

3. **Configure Package**

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

   Initially two files may require alteration.

   1. The startup file, possibly to set the local WASD_CONFIG_GMT logical (for systems not supporting DTSS (e.g. DECnet-Plus)). Consider using the STARTUP_LOCAL.COM file for other site-specific requirements (Section 3.3).
   2. The only configuration that should require immediate attention will be to the mapping rules (Chapter 12).

   More generally server runtime configuration involves the considerations discussed in Section 4.2 along with the following aspects:

   - Configuring the HTTP server run-time characteristics (Chapter 4).
   - Mapping request paths to the VMS file system, and to other things such as scripts (Chapter 12).
   - Customizing some messages (or all if non-English language environment) (Chapter 10).
   - Establishing an authentication and authorization environment.

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 the run-time log, look in the WASD_ROOT:[LOG_SERVER] directory.

   Remember, the basic installation's integrity can always be checked as described in Chapter 2, Section 2.10. 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 2.17.

Chapter 2
Installation and Update

The WASD package is distributed as ZIP archives.

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

```
$ UNZIP -v
```

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

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

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

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

The archive will have the structure:

```
Archive:  SYS$SYSDEVICE:[WASD]WASD1100.ZIP;1

 WASD VMS Web Services, Copyright (C) 1996-2016 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 3, or any later version.
 http://www.gnu.org/licenses/gpl.txt

  *  Full release of v11.0.0 (May 2016)
     ************************************************
     *** CONTAINS SOURCE FILES, DOCUMENTATION, ETC. ***
     ************************************************
    Package must be built using INSTALL or UPDATE as described below.

  *  To install files:
     $ SET DEFAULT device:[000000]
     $ UNZIP device:[dir]WASD1100.ZIP
```

```
 *  To build/link images use the appropriate one of:
    $ @device:[WASD_ROOT]INSTALL
    $ @WASD_ROOT:[000000]UPDATE

8< snip 8<
VMS file attributes saved ... use UnZip 5.2+ on OpenVMS

Archive created 1-MAY-2016

 Length     Date   Time    Name
--------    ----   ----    ----
       0  04-17-16 00:50   wasd_root/axp-bin/
       0  04-17-16 00:50   wasd_root/axp/
       0  04-17-16 00:50   wasd_root/cgi-bin/
       0  04-17-16 00:51   wasd_root/doc/
       0  04-17-16 00:51   wasd_root/example/
       0  04-17-16 00:51   wasd_root/exercise/
    2734  03-06-03 17:20   wasd_root/favicon.ico
8< snip 8<
     270  11-05-09 06:59   wasd_root/startup/readme.html
     426  11-05-09 06:38   wasd_root/vax-bin/readme.html
     452  11-05-09 06:18   wasd_root/vax/readme.html
--------                   -------
20288784                   919 files
```

2.1 Package UNZIP

The archive contains the complete directory tree. Hence it is necessary to SET DEFAULT into the top-level directory of the volume the package is to be installed on.

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

Updating From v9.3 or Earlier

Before UNZIPing the v11 package and when updating an existing v9.3 or earlier installation the **root directory must be renamed from HT_ROOT.DIR to WASD_ROOT.DIR**. The v10 and later packages use [WASD_ROOT] as the top-level directory in line with other naming schema changes employing "WASD". Remember to modify local startup procedures in-line with this new top-level directory name. Also note that the v11 package is not suitable for updating from existing v8.0 or earlier installation.

Source Archive, Object Module Archives

The complete package, source code, documentation, examples, etc., is provided in a single main archive. Installation and other build procedures allow the entire package to be compiled and linked from this if prefered. This requires a later version of DEC C (preferably v6.*n* or greater).

In addition, for those unable or not wishing to fully build the distribution, three other platform-specific archives are available, AXP (Alpha) IA64 (Itanium) and VAX, containing a complete set of object modules, allowing the package to be built via a link operation only.

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

```
$ UNZIP device:[dir]archive-AXP.ZIP
$ UNZIP device:[dir]archive-IA64.ZIP
$ UNZIP device:[dir]archive-VAX.ZIP
```

Note

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

WASD OpenSSL Archive

Building an SSL-capable version of the server is a common requirement. WASD SSL is discussed in detail in "WASD Web Services - Features and Facilities" and if using the WASD SSL package it is also possible to install (or update) that package after UNZIPing the primary archive and optional object module(s). As noted in the above SSL section, the server can also be built against an existing VMS SSL product and an existing OpenSSL installation.

Note

The WASD OpenSSL kit is designed as an update to an existing WASD installation and so expects to be UNZIPed under the root directory. Note the use of the "-d" switch in the following example.

```
$ UNZIP -d [.WASD_ROOT] device:[dir]OPENSSLWASDnnn-arch.ZIP
```

Existing Installations

When installing an archive as an update to an existing installation consider the following.

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

  ```
  $ BACKUP WASD_ROOT:[000000...] location:WASDROOT.BCK/SAVE/VERIFY

  $ ZIP "-V" location:WASDROOT.ZIP device:[WASD_ROOT...]*.*
  $ ZIP "-T" location:WASDROOT.ZIP
  ```

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

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

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

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

Multiple Installations

Multiple, independent installations and builds of WASD are supported. See Section 2.13 later in this chapter.

2.2 ODS-5 Volumes

The WASD package can be installed on and used from ODS-5 (extended file specification) volumes. Note that the installation procedures and file system organisation of the package tree has been designed for ODS-2 compliance. (Of course the issue of installing WASD on an ODS-5 volume is completely separate from the ability to serve the contents of an ODS-5 volume!)

2.3 Accessible Volume

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

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

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

2.4 Package Directory Structure

The package directories and content are organised as follows. Note that only some of these can be accessed by the server account (and therefore seen in server-generated directory listings) due to directory and file protections (Section 5.2).

Package Directory Structure

Directory	Description
[AXP-BIN]	Alpha executable script files
[AXP]	Alpha build and utility area
[CGI-BIN]	architecture-neutral script files
[DOC]	package documentation
[EXAMPLE]	package examples

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

2.5 TCP/IP Infrastructure

The WASD installation assumes that the system's TCP/IP infrastructure is correctly installed and configured, and is operating normally. For example, it is not unknown for a freshly built system to experience host name resolution problems preventing its own host name from being resolved and making even elementary server startup impossible.

2.6 SYSUAF and RIGHTSLIST WARNING!

The WASD installation procedure does, and to a lesser degree the update procedure can, **make additions and/or modifications to SYSUAF.DAT and RIGHTLIST.DAT**, for default server and scripting accounts and to facilitate their access to the package directory tree.

Also, **when the server image begins execution it may add an identifier**, required for script process management, to RIGHTSLIST.DAT.

These behaviours must be considered in site environments where such changes are prohibited or closely controlled.

2.7 Installation DCL Procedure

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

After UNZIPing the package do the following:

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

It performs the following tasks:

1. **Build Executables -** Either compile sources and link, or just link package object code to produce images for the local version of VMS. If the system has a suitable SSL toolkit the installer is requested whether an SSL enabled version be built. Note the possible UDFSYM described in Section 2.9.
2. **Check Package -** Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package (Section 2.10).
3. **Create Server and Scripting Accounts -** Create two, independent accounts, one for executing the server, the other for executing scripts (Section 3.1). If quotas are enabled on the target disk provides an ambit allocation for these accounts. Review this at some stage.
4. **Set Package Security -** This section traverses the newly installed tree and sets all package directories and files to required levels of access. For directory settings see Section 5.2.
5. **Copy Support and Configuration Files -** Copy the example server support and configuration files (Section 3.3).
6. **Install Scripts -** Selectively copy groups of scripts from package build directories into the scripting directories.

Support files to consider when customizing startup, etc. (see Section 3.3 for further detail):

STARTUP.COM
STARTUP_LOCAL.COM
STARTUP_SERVER.COM

2.8 Update DCL Procedure

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

Updating From v9.3 or Earlier

Before UNZIPing the v11 package and when updating an existing v9.3 or earlier installation the current **root directory must be renamed from HT_ROOT.DIR to WASD_ROOT.DIR**. The v10 and later packages use [WASD_ROOT] as the top-level directory in line with other naming schema changes employing "WASD". Remember to modify local startup procedures in-line with this new top-level directory name. Also

note that the v11 package is not suitable for updating from an existing v8.0 or earlier installation.

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

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

After UNZIPing the updated package do the following:

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

It provides the following functions:

1. **Build Executables -** Either compile sources and link, or just link package object code to produce images for the local version of VMS. If the system has a suitable SSL toolkit the installer is requested whether an SSL enabled version be built. Note the possible UDFSYM described in Section 2.9.
2. **Server Quick-Check -** Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package (Section 2.10).
3. **Server Support/Configuration Files -** Copies changed example HTTP server configuration and support files from the [EXAMPLE] directory to the [HTTP$SERVER], [LOCAL] and [STARTUP] directories.
4. **Update Scripts -** Selectively copy groups of scripts from package build directories into the scripting directories.
5. **Reapply Package Security -** This section traverses the updated tree and sets all package directories and files to required levels of access. For directory settings see Section 5.2.
6. **Post-Update Cleanup -** Prompts for permission to execute the post-update procedure described below.
7. **Purge Files -** Prompts for permission to purge the entire WASD_ROOT:[000000...] tree.

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

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

2.9 %LINK-I-UDFSYM

Linking the server code against older versions of OpenSSL (less likely) or the HP SSL product (more likely, V1.4 for instance) will result in the reporting of one, two or three undefined symbols (usually one or two as shown below).

```
%LINK-W-NUDFSYMS, 1 undefined symbol:
%LINK-I-UDFSYM, SSL_GET_SERVERNAME
```

```
%LINK-W-NUDFSYMS, 2 undefined symbols:
%LINK-I-UDFSYM, TLSV1_1_METHOD
%LINK-I-UDFSYM, TLSV1_2_METHOD
```

Any of these three reports may safely be ignored as the server is designed to detect the absence and disable the related functionality.

2.10 Quick-Check

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

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

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

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

Use a browser to access either of the "%HTTPD-I-SERVICE"s when the server
starts.  (There will be one for a standard service and another for SSL.)

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

Password (for demo authentication)? []: anyoldpassword

%DCL-S-SPAWNED, process SYSTEM_61053 spawned
%DCL-S-ATTACHED, terminal now attached to process SYSTEM_61053
%HTTPD-I-SOFTWAREID, HTTPd-WASD/11.0.0 OpenVMS/AXP SSL
WASD VMS Web Services, Copyright (C) 1996-2016 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 3, or any later version.
http://www.gnu.org/licenses/gpl.txt
%HTTPD-I-STARTUP, 01-MAY-2016 22:39:04
%HTTPD-I-ALIGN, start collecting alignment faults (64kB,128,0xFFFFFFF0)
%HTTPD-I-WASD_ROOT, $1$DKA0:[WASD_ROOT]
%HTTPD-I-ENVIRONMENT, 0
%HTTPD-I-SYSTEM, Digital Personal WorkStation  VMS V8.3
%HTTPD-W-SYSPRV, operating with implicit SYSPRV (UIC group 1)
%HTTPD-I-TCPIP, HP TCPIP$IPC_SHR V5.7-ECO1 (21-MAY-2010 14:44:46.64)
%HTTPD-I-MODE, INTERACTIVE
%HTTPD-I-ODS5, supported by Alpha VMS V8.3
%HTTPD-I-GMT, +09:30
%HTTPD-I-INSTANCE, supervisor
%HTTPD-I-GZIP, using GNV$LIBZSHR32 V1.2.8
%HTTPD-I-GBLSEC, created global section of 16 page(let)s
%HTTPD-I-INSTANCE, 1 process
%HTTPD-I-SSL, OpenSSL 1.0.2g  1 Mar 2016
-SSL-I-PROTOCOL, TLSv1,TLSv1.1,TLSv1.2
-SSL-I-OPTIONS, 0x80510BFF
-SSL-I-SNI, Server Name Indication enabled
-SSL-I-DH, ephemeral DH param 1024,2048 bits
%HTTPD-I-HTTP2, enabled
%HTTPD-W-HTTP2, network/mailbox read buffer size increased to 16384 bytes
```

```
%HTTPD-I-INSTANCE, process name WASD:7080
%HTTPD-I-WEBDAV, disabled
%HTTPD-W-AUTH, 1 informational, 1 warning, 0 errors at load
1.w PROMISCUOUS authenticating any username with specified password!
2.i Cache for 32 records of 768 bytes in local storage of 49 page(let)s
%HTTPD-W-MAP, 1 informational, 0 warning, 0 errors at load
1.i ODS-5 processing enabled
%HTTPD-I-PROXYVERIFY, for 32 records in local storage of 14 page(let)s
%HTTPD-I-SCRIPTING, as HTTP$NOBODY
%HTTPD-I-DCL, subprocess scripting
%HTTPD-I-ACTIVITY, created global section of 1312 page(let)s
%HTTPD-I-SERVICE, http://klaatu.private:7080
%HTTPD-I-SERVICE, https://klaatu.private:7443
%HTTPD-I-SSL, klaatu.private:7443
%HTTPD-I-DEMO, demonstration mode
1.i subprocess scripting
2.i promiscuous authentication
3.i directory access control files ignored
4.i [DirAccess] enabled
5.i [DirMetaInfo] enabled
6.i [DirWildcard] enabled
7.i [Logging] disabled
8.i [ReportBasicOnly] disabled
9.i [ReportMetaInfo] enabled
%HTTPD-I-BEGIN, 01-MAY-2016 22:39:05, accepting requests
```

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

```
        -
     /-- / \
    /W A S D\     Welcome to "WASD VMS Web Services" version 11.0
Empowered by VMS
    \/---\  /
         --
```

Note

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

2.11 “Clone” Procedure

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

2.12 Re-Linking

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

```
%DCL-W-ACTIMAGE, error activating image WHATEVER
-CLI-E-IMGNAME, image file DKA0:[SYS0.SYSCOMMON.][SYSLIB]WHATEVER_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 2.8.

2.13 Multiple Installations

It is possible, and often useful, to build another WASD on a system with an existing and/or running installation. One purpose might be to maintain the previous version as a fallback in case of unexpected problems when migrating to a more recent version. Another, to maintain multiple releases for regression testing.

The general process is as follows:

- A clash with any existing [WASD_ROOT] directory must be avoided.
  Using the "-d" switch, UNZIP into a working directory using any unique name (BLAH in this example).

  ```
  $ SET DEFAULT device:[000000]
  $ UNZIP -d [.BLAH] device:[dir]archive.ZIP
  ```

  Do the same with object module archive(s) if required.

- For the WASD OpenSSL package the WASD_ROOT portion of the tree must additionally be specified.

  ```
  $ UNZIP -d [.BLAH.WASD_ROOT] device:[dir]OPENSSLWASDnnn-arch.ZIP
  ```

- Rename the WASD root directory into the current directory using a representative name (appending the version number is suggested) and then delete the working directory.

  ```
  $ RENAME [.BLAH]WASD_ROOT.DIR []WASD_ROOT_nnnn.DIR
  $ DELETE BLAH.DIR;*
  ```

- Move into the just renamed directory and build using the parameter INSTALL.
  The build is (always) performed using locally defined logical names.

  ```
  $ SET DEFAULT [WASD_ROOT_nnnn]
  $ @INSTALL INSTALL
  ```

  The INSTALL parameter overrides the install check and advisory message otherwise generated:

  ```
  ****************************************
  *   "WASD_ROOT" LOGICAL NAME DETECTED.  *
  *  THIS DOES NOT LOOK LIKE AN INSTALL!  *
  ****************************************
  ```

- As appropriate, copy configuration and other files from the current WASD installation to the new.
  Check release notes for any variants.

  ```
  $ COPY WASD_ROOT:[LOCAL]*.* [WASD_ROOT_nnnn.LOCAL]
  ```

  Other site-specific localisations similarly may need to be copied or otherwise reproduced. For example, server or scripting account LOGIN.COM, scripts, etc.

To move the running WASD environment from one installation to another:

- Shut down the currently running server.

  ```
  $ HTTPD/DO=EXIT=NOW
  ```

- Start the desired version of WASD from its file-system location.

  ```
  $ @device:[WASD_ROOT_nnnn.STARTUP]STARTUP.COM
  ```

  WASD logical names and environment will reflect the particular WASD root directory. Site-specific elements in the startup might need to be similarly flexible.

2.14 VMS 6.*n*

As of WASD v10.1 the minimum supported version for build and operation is VMS V7.0. Had to drag ourselves into the mid-1990s at some stage!

2.15 VMS 5.5-*n*

WASD does not build or run under VMS 5.5-2 or earlier.

2.16 Local Setup Suggestions

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

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

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

2.17 Reporting Problems

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

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

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

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

The WATCH facility is often a powerful tool for problem investigation. It is also very useful when supplying details during problem resolution. **When supplying WATCH output as part of a problem report please ZIP the file and include it an an e-mail attachment.** Mailers often mangle the report format making it difficult to interpret.

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

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

Chapter 3

Server Account and Environment

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

Process Quotas!

Server process quotas must be sufficient to support the expected traffic load. BYTLM in particular, and then BIOLM, DIOL, FILLM and PGFLQUO, are all considerations.

Symptoms of insufficient process quotas include:

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

A general rule is more is better, after all, it will only use as much as it needs! To assist with setting a reasonable BYTLM quota the WATCH report (see "WASD Web Services - Features and Facilities") provides some feedback on server BYTLM usage.

TCP/IP Agent Resources!

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

Later versions of TCP/IP Services for OpenVMS seem to have large default values for socket send and receive buffers. MultiNet and TCPware are reported to improve transfer of large responses by increasing low default values for send buffer size. The WASD global configuration directives [SocketSizeRcvBuf] and [SocketSizeSndBuf] allow default values to be adjusted. WATCH can be used to report network connection buffer values.

3.1 VMS Server Account

The following provides a guide to the account.

```
Username: HTTP$SERVER                       Owner:  WASD Server
Account:  HTTPD                             UIC:    [077,001] ([HTTP$SERVER])
CLI:      DCL                               Tables: DCLTABLES
Default:  WASD_ROOT:[HTTP$SERVER]
LGICMD:   LOGIN
Flags:  Restricted DisNewMail
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
Primary   000000000011111111112222  Secondary 000000000011111111112222
Day Hours 012345678901234567890123  Day Hours 012345678901234567890123
Network:  ##### Full access ######            ##### Full access ######
Batch:    ##### Full access ######            ##### Full access ######
Local:    -----  No access  ------             -----  No access  ------
Dialup:   -----  No access  ------             -----  No access  ------
Remote:   -----  No access  ------             -----  No access  ------
Expiration:           (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:      (pre-expired)
Last Login:           (none) (interactive), 11-MAY-1995 08:44 (non-interactive)
Maxjobs:         0  Fillm:       300  Bytlm:      5000000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:           0
Maxdetach:       0  BIOlm:      2048  JTquota:       1024
Prclm:         100  DIOlm:      1024  WSdef:         1000
Prio:            4  ASTlm:      2000  WSquo:         5000
Queprio:         0  TQElm:       100  WSextent:     20000
CPU:        (none)  Enqlm:       256  Pgflquo:     500000
Authorized Privileges:
  NETMBX    TMPMBX
Default Privileges:
  NETMBX    TMPMBX
```

3.2 VMS Scripting Account

The following provides a guide to the account.

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

3.3 Account Support Files

Note

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

online Web link

HTTPd Executables

Two server executables can be built by the package.

- **HTTPD.EXE -** basic server
- **HTTPD_SSL.EXE -** SSL-enabled server.

Privileged Image

As the HTTP$SERVER account should be completely unprivileged, and the HTTPd image requires ALTPRI, CMKRNL, DETACH, NETMBX, TMPMBX, PRMGBL, PRMMBX, PSWAPM, SECURITY, SHMEM (VAX only), SYSGBL, SYSLCK, SYSNAM, SYSPRV and WORLD privileges (see the “WASD_ROOT:[SRC.HTTPD]READMORE.TXT” document for a description of how and why the server uses these privileges).

It is installed using a command similar to the following:

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

STARTUP.COM

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

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

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

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

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

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

On VAX platforms prior to VMS V6.2 the startup uses a system batch queue. By default SYS$BATCH is used. An alternate queue can be specified.

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

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

See WASD_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. For example, a STARTUP.COM defined logical name could be modified here before the server proper is actually started.

See WASD_ROOT:[EXAMPLE]STARTUP_LOCAL.COM

STARTUP_SERVER.COM

This procedure serves two purposes.

1. Server startup:
   - If on VAX VMS V6.0 or V6.1 it is submitted to the SYS$BATCH queue during startup. The batch portion creates a detached process, which then again uses this procedure as input, supporting the executing HTTPd.
   - With more modern versions and architectures of VMS the procedure becomes SYS$COMMAND for a detached process created directly during the execution of STARTUP.COM.
2. The procedure then controls the activation of the HTTPd executable image during server restarts and exits.

See WASD_ROOT:[EXAMPLE]STARTUP_SERVER.COM

It is recommended to pass server startup command-line parameters using the WASD_SERVER_STARTUP logical name that this procedure checks for and uses if present. If this is defined the contents are applied to the server image when executed. It can be explicitly defined before WASD startup.

```
$ DEFINE /SYSTEM /EXECUTIVE WASD_STARTUP_SERVER "/SYSUAF=ID"
$ @DKA0:[WASD_ROOT.STARTUP]STARTUP
```

The value can also be passed to the main startup procedure in a symbol. The startup procedure then defines a system logical name with that value (note that any quotes used must be escaped).

```
$ WASD_DECNET = 1
$ WASD_SSL = 1
$ WASD_SSL_CERTIFICATE = "WASD_ROOT:[LOCAL]ALPHA.PEM"
$ WASD_STARTUP = "/SYSUAF=ID"
$ @DKA0:[WASD_ROOT.STARTUP]STARTUP
```

It can also be manually redefined at any time and the server restarted to apply different startup parameters to the running server.

```
$ DEFINE /SYSTEM /EXECUTIVE WASD_STARTUP_SERVER "/SYSUAF=(SSL,ID)"
$ HTTPD /DO=RESTART=NOW
```

3.4 Global Pages/Sections

Various accounting, cache and other shared data used by the server is provided by shared global memory. These requires one permananet global section (SYSGEN parameter GBLSECTIONS) and a number of permanent global pages (SYSGEN parameter GBLPAGES) per item. The number of items varies depending on configuration.

Global Sections

Item	Description	Usage
Accounting	Accumulates various data provided to the Server Administration Statistics report and the HTTPMON utility	required
Activity	Provides data to the Server Administration Activity Report graph	required
Authentication	When multiple WASD Instances are configured provides a shared authentication cache	optional
Proxy Verification	When multiple WASD Instances are configured provides an shared proxy verification cache	optional
SSL Session Cache	When SSL is used and multiple WASD Instances are configured provides a shared SSL session cache	optional

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

As permanent, system-accessible global sections are deployed it may be necessary to explicitly delete them after ad hoc server experimentation, etc. (Section 3.6). The startup qualifier /GBLSEC=NOPERM disables the creation of permanent global sections eliminating this requirement.

3.5 Logical Names

WASD version 10 uses an independent logical name table (something previous versions did not, see Section 3.5.1 below) and a different logical naming schema to earlier versions.

The following logical names are used in the operation of the package. These are usually created by STARTUP.COM during server startup.

Package Logical Names

Logical Name	Table	Description	Pre-v10 Equivalent
CGI-BIN	WASD	(Hyphen) 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.	same
CGI_BIN	WASD	Directory containing architecture-neutral script files.	same
CGI_EXE	WASD	Directory containing architecture-specific script executables.	same
HT_EXE	WASD	Pre-v10.0 backward compatibility for WASD_EXE.	same

Logical Name	Table	Description	Pre-v10 Equivalent
HT_LOGS	WASD	Pre-v10.0 backward compatibility for WASD_LOG.	same
HT_ROOT	SYSTEM	Pre-v10.0 backward compatibility for WASD_ROOT.	same
HT_SCRATCH	WASD	Pre-v10.0 backward compatibility for WASD_SCRATCH.	same
WASD_AXP	WASD	Directory containing Alpha executable images (WASD_ROOT:[AXP]).	HT_AXP **
WASD_AUTH	WASD	Directory containing authentication/authorization databases (files, (WASD_ROOT:[LOCAL])).	none
WASD_CGI_AXP	WASD	Directory containing Alpha script executables (WASD_ROOT:[AXP-BIN]).	CGI_AXP
WASD_CGI_IA64	WASD	Directory containing Itanium script executables (WASD_ROOT:[IA64-BIN]).	CGI_IA64
WASD_CGI_VAX	WASD	Directory containing VAX script executables (WASD_ROOT:[VAX-BIN]).	CGI_VAX
WASD_CONFIG	WASD	Location of the configuration files. Can be defined as a search list.	none
WASD_CONFIG_AUTH	WASD	Location of the authentication/authorization configuration file.	HTTPD$AUTH
WASD_CONFIG_GLOBAL	WASD	Location of the configuration file.	HTTPD$CONFIG
WASD_CONFIG_MAP	WASD	Location of the mapping rule file.	HTTPD$MAP
WASD_CONFIG_MSG	WASD	Location of the message file.	HTTPD$MSG
WASD_CONFIG_SERVICE	WASD	Location of the optional service (virtual host) configuration file.	HTTPD$SERVICE
WASD_DECNET_CGI_OBJECT	SYSTEM	Locates the supporting DCL procedure. DECnet objects are system-global.	none
WASD_DECNET_OSU_OBJECT	SYSTEM	Locates the supporting DCL procedure. DECnet objects are system-global.	none
WASD_EXE	WASD	Directory containing the executable images.	HT_EXE **

Logical Name	Table	Description	Pre-v10 Equivalent
WASD_FILE_DEV[*n*]	SYSTEM	Locates the DCL procedure that will integrate the specified environment's logical name table into the processes' LNM$FILE_DEV (see above).	none
WASD_GMT	WASD	Offset from GMT (e.g. "+10:30", "-01:15") For systems supporting DTSS (e.g. DECnet-Plus) this logical may be left undefined, with server time being calculated using the SYS$TIMEZONE_DIFFERENTIAL logical.	HTTPD$GMT
WASD_IA64	WASD	Directory containing Itanium executable images.	HT_IA64
WASD_LOG	WASD	If logging is enabled and no log file name specified on the command line, this logical must be defined to locate the file. When a logging period is in use this logical need only contain the directory used to store the logs.	HT_LOG
WASD_LOGS	WASD	Optional definition, for convenient log file specification.	HT_LOGS **
WASD_ROOT	SYSTEM	Location of WASD Web Services directory tree, as a concealed device.	HT_ROOT **
WASD_SCRATCH	WASD	Location of an optional directory that scripts can use for temporary storage. Must be read+write+delete accessible to the server account. The WASD_CONFIG_GLOBAL [DclCleanupScratchMinutesMax] directive controls whether automatic cleanup scans of this area delete any files that are older than [DclCleanupScratchMinutesOld].	HT_SCRATCH **
WASD_SITELOG	WASD	Location of the optional plain-text site log file.	HTTPD$SITELOG
WASD_SSL_CAFILE	WASD	When using the SSL executable this logical locates the optional Certificate Authority list file.	HTTPD$SSL_CAFILE
WASD_SSL_CERT	WASD	When using the SSL executable this logical locates the default certificate.	HTTPD$SSL_CERT
WASD_SERVER_LOGS	WASD	Location of the server process logs.	HT_SERVER_LOGS **

Logical Name	Table	Description	Pre-v10 Equivalent
WASD_STARTUP_ SERVER	WASD	Used to pass parameters to the server image startup command line.	HTTPD_STARTUP_ SERVER
WASD_VAX	WASD	Directory containing VAX executable images.	HT_VAX **
			***provided for back-ward compatibility*

3.5.1 WASD Name Table

In an effort to localise WASD-related logical names and avoid polluting the SYSTEM logical name table WASD version 10 creates it's own world-readable, system-writable name table, and adds it to LNM$SYSTEM_DIRECTORY.

```
$ SHOW LOGICAL WASD_TABLE/TABLE=LNM$SYSTEM_DIRECTORY
   "WASD_TABLE" [table] = "" (LNM$SYSTEM_DIRECTORY)
```

WASD logical names are then defined in that table leaving the SYSTEM table with just a few essential names.

```
$ SHOW LOGICAL CGI*,HT*,WASD*,WWW*

(LNM$PROCESS_TABLE)

(LNM$JOB_81E3D580)

(WASD_TABLE)
  "CGI-BIN" = "DKA0:[WASD_ROOT.CGI-BIN.]"
        = "DKA0:[WASD_ROOT.AXP-BIN.]"
  "CGI_BIN" = "WASD_ROOT:[CGI-BIN]"
  "CGI_EXE" = "WASD_ROOT:[AXP-BIN]"
  "HTBIN" = "CGI-BIN:[000000]"
  "HT_CACHE_ROOT" = "DKA0:[HT_CACHE.]"
  "HT_EXE" = "WASD_ROOT:[AXP]"
  "HT_LOGS" = "WASD_ROOT:[LOG]"
  "HT_SCRATCH" = "WASD_ROOT:[SCRATCH]"
  "WASD_AUTH" = "WASD_ROOT:[LOCAL]"
  "WASD_AXP" = "WASD_ROOT:[AXP]"
  "WASD_CACHE_ROOT" = "DKA0:[HT_CACHE.]"
  "WASD_CGILIBSHR32" = "CGI_EXE:CGILIBSHR32.EXE"
  "WASD_CGI_AXP" = "WASD_ROOT:[AXP-BIN]"
  "WASD_CGI_BIN" = "WASD_ROOT:[CGI-BIN]"
  "WASD_CGI_EXE" = "WASD_ROOT:[AXP-BIN]"
  "WASD_CGI_IA64" = "WASD_ROOT:[IA64-BIN]"
  "WASD_CGI_VAX" = "WASD_ROOT:[VAX-BIN]"
  "WASD_CONFIG" = "WASD_ROOT:[LOCAL]"
  "WASD_CONFIG_AUTH" = "WASD_CONFIG:HTTPD$AUTH.CONF"
  "WASD_CONFIG_GLOBAL" = "WASD_CONFIG:HTTPD$CONFIG.CONF"
  "WASD_CONFIG_MAP" = "WASD_CONFIG:HTTPD$MAP.CONF"
  "WASD_CONFIG_MSG" = "WASD_CONFIG:HTTPD$MSG.CONF"
  "WASD_CONFIG_SERVICE" = "WASD_CONFIG:HTTPD$SERVICE.CONF"
  "WASD_EXE" = "WASD_ROOT:[AXP]"
  "WASD_HTTPD_EXE" = "WASD_EXE:HTTPD_SSL.EXE"
  "WASD_IA64" = "WASD_ROOT:[IA64]"
  "WASD_JAVA" = "WASD_ROOT:[JAVA]"
```

```
  "WASD_LOCAL" = "WASD_ROOT:[LOCAL]"
  "WASD_LOGS" = "WASD_ROOT:[LOG]"
  "WASD_SCRATCH" = "WASD_ROOT:[SCRATCH]"
  "WASD_SCRIPT" = "WASD_ROOT:[SCRIPT]"
  "WASD_SCRIPT_LOCAL" = "WASD_ROOT:[SCRIPT_LOCAL]"
  "WASD_SERVER_LOGS" = "WASD_ROOT:[LOG_SERVER]"
  "WASD_SSL_CAFILE" = "WASD_CONFIG:CA-BUNDLE_CRT.TXT"
  "WASD_SSL_CERT" = "WASD_CONFIG:HTTPD.PEM"
  "WASD_STARTUP" = "WASD_ROOT:[STARTUP]"
  "WASD_STARTUP_SERVER" = "/SYSUAF=(ID,SSL)/PERSONA=RELAXED/PROFILE"
  "WASD_VAX" = "WASD_ROOT:[VAX]"
  "WWW_ROOT" = "DKA0:[WASD_ROOT.SRC.OSU]"
  "WWW_SCRIPT_MAX_REUSE" = "999"

(LNM$GROUP_000001)

(LNM$SYSTEM_TABLE)

  "HT_ROOT" = "DKA0:[WASD_ROOT.]"
  "WASD_DECNET_CGI_OBJECT" = "DKA0:[WASD_ROOT.CGI-BIN]CGIWASD.COM"
  "WASD_DECNET_OSU_OBJECT" = "DKA0:[WASD_ROOT.CGI-BIN]WWWEXEC.COM"
  "WASD_FILE_DEV" = "DKA0:[WASD_ROOT]WASD_FILE_DEV.COM"
  "WASD_ROOT" = "DKA0:[WASD_ROOT.]"

(LNM$SYSCLUSTER_TABLE)

(DECW$LOGICAL_NAMES)
```

As can be seen the number of LNM$SYSTEM_TABLE names is small, five in this example (though it can vary). Logical name WASD_FILE_DEV locates a procedure to insert the WASD_TABLE into a process' LNM$FILE_DEV to make the table names available. Until that is done they are not visible without an explicit /TABLE=WASD_TABLE. The server automatically uses the procedure for itself and scripting processes. Site admins can simply

```
$ @WASD_FILE_DEV
```

at the command-line or in their LOGIN.COM to have it done for their interactive session(s). This procedure location is variable within the file-system and needs to be located and accessed without initially knowing that location.

The WASD_ROOT logical provides a convenient, global logical location for the primary (default) WASD environment. HT_ROOT is used to provide pre-v10 backward-compatibility with existing sites. (If yours does not need the name you can deassign it during server startup.)

The WASD_DECNET_CGI_OBJECT and WASD_DECNET_OSU_OBJECT names provide global locations for the two DECnet scripting environments. These logicals are defined when a site uses the [STARTUP]STARTUP_DECNET.COM procedure. It is necessary to provide a global location for these with multiple WASD environments because DECnet objects are global entities. The one object must provide an infrastructure for potentially multiple WASD environments.

Other SYSTEM logical names, WASD_TABLE*n* name tables, and WASD_FILE_DEV*n* logical names are used for non-primary WASD environments (see "WASD Web Services - Features and Facilities").

3.5.2 Pre-v10

The server code accepts both the v10 or later and pre-v10 schemas. If it cannot find a v10 logical name it attempts to use a pre-v10 logical name. This has been provided in an effort to make the transition as seamless as possible for existing sites. In addition the revised startup procedures configure and use WASD_TABLE but can be directed to use the SYSTEM table by STARTUP.COM being provided a WASD_TABLE=0 parameter (see STARTUP.COM).

```
$ WASD_TABLE = 0
$ @DKA0:[WASD_ROOT.STARTUP]STARTUP.COM
```

3.6 Server Startup

When starting up the server several characteristics of the server may be specified using qualifiers on the command line. If not specified appropriate defaults are employed. For recommended methods of passing parameters to the executable at server startup see STARTUP_SERVER.COM. For clarity some esoteric and legacy qualifiers and parameters are not listed in this table.

Server Image Command-Line Parameters

Parameter/Qualifier	Description
/ALL[=integer]	Has two roles. When starting a server up assigns that server to a specific, non-default WASD environment (see /ENVIRONMENT) When using the server control /DO= using /ALL specifies to do the action to all servers in that particular environment.
/AUTHORIZATION=..	Control authentication and authorisation behaviour. See “WASD Web Services - Features and Facilities”
/CGI_PREFIX=	The prefix to the CGI symbol names created for a script (defaults to “WWW_”). See “WASD Web Services - Scripting”
/CLUSTER	Apply control /DO= to all instances in a cluster (default is to current node instance(s) only).
/DETACH=	This qualifier allows a DCL procedure to be specified as input to a directly detached process (in conjunction with /USER).
/DO=	Command to be performed by the executing server.
/ENVIRONMENT=	Integer indicating in which environment this server is executing
/GBLSEC=DELETE	Allows a monitor-associated permanent global section to be explicitly deleted. When a server starts it creates system-accessible, permanent global sections in which to store accounting and request data. As this is permanent it would be possible for a site, perhaps experimenting with servers over a range of ports, to consume significant amounts of global pages and sections. This qualifier allows such sections to be deleted.

Parameter/Qualifier	Description
/GBLSEC=NOPERM	Disables the creation of permanent global sections. They are automatically deleted when the server image exits.
/[NO]LOG[=name]	Either disables logging (overrides configuration directive), or enables logging and optionally specifies the log file name (also see section Section 3.5, 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.
/NETWORK	Run the server and any scripting processes as NETWORK mode rather than the default detached OTHER mode.
/NOTE=string	Annotate the server process log with the specified string. Intended to assist auditing server events such as restarts, maaping reloads and the like.
/PERSONA[=..]	Enables and controls detached process scripting. See WASD Web Services - Scripting
/PRIORITY=	Server process priority (default is 4).
/[NO]PROFILE	Allows SYSUAF-authenticated username security profiles to be used for file access.
/PROMISCUOUS[=password]	Server will accept any authentication username/password pair (used for testing, demonstrations, etc.)
/PROXY=string	Allows proxy maintainance activities to be executed from the command line (e.g. from batch jobs, etc.).
/SCRIPT=AS=username	Specifies the username of the default scripting account.
/SERVICE=	Comma-separated, list of server services (overrides the [Service] configuration parameter).
/SOFTWARE=	An arbitrary string that can be used to override the server software identification (i.e. “HTTPd-WASD/10.4.0 OpenVMS/AXP SSL”).
/[NO]SSL[=..]	Controls Secure Sockets Layer protocol behaviour. See “WASD Web Services - Features and Facilities”
/[NO]SYSUAF[=..]	Controls VMS (SYSUAF) authentication/authorisation behaviour. See “WASD Web Services - Features and Facilities”
/USER=username	For VMS 6.2 and later this qualifier allows the /DETACH qualifier to directly create a detached process executing as the specified username.
/VALBLK[=16 | 64]	For server to (try) to use either pre-VMS V8.2 16 byte lock value block or the VMS V8.2 and later 64 byte lock value block.
/VERSION	Displays the executable’s version string and the copyright notice.

Parameter/Qualifier	Description
/[NO]WATCH[=..]	Controls the use of the WATCH reporting facility. See “WASD Web Services - Features and Facilities”

Chapter 4
Configuration Considerations

WASD has a global configuration, which applies characteristics to the entire running server, as well as per-service (virtual server) and conditional configuration, which applies characteristics or behaviours to specific requests. All configuration is provided via files located by logical names.

Configuration Files

Name	Scope	Description
WASD_CONFIG_AUTH	loadable	request authorization control
WASD_CONFIG_GLOBAL	global	global server configuration
WASD_CONFIG_MAP	loadable	request processing control
WASD_CONFIG_MSG	global	provides server messages
WASD_CONFIG_SERVICE	global	specifies services (virtual servers)

Simple editing of these files change the configuration. Comment lines may be included by prefixing them with the hash ("#") character. Comment lines prefixed with a quote and then a hash ("!#") are displayed in Server Admin reports and are WATCHable during rule proceessing. Configuration file directives are not case-sensitive. Any changes to global configuration file can only be enabled by restarting the HTTPd process using the following command on the server system.

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

Changes to request mapping or authorization configuration files also can be dynamically reloaded into the running server using the administration command-line interface.

```
$ HTTPD /DO=MAP=LOAD
$ HTTPD /DO=AUTH=LOAD
```

Changes to configuration files can be validated at the command-line before reload or restart. This detects and reports any syntactical and fatal configuration errors but of course cannot check the *intent* of the rules.

```
$ HTTPD /DO=AUTH=CHECK
$ HTTPD /DO=CONFIG=CHECK
$ HTTPD /DO=GLOBAL=CHECK
$ HTTPD /DO=MAP=CHECK
$ HTTPD /DO=MSG=CHECK
$ HTTPD /DO=SERVICE=CHECK
```

The *config* check sequentially processes each of the *authorization*, *global*, *mapping*, *message* and *service* configuration files.

If additional server startup qualifiers are required to enable specific configuration features then these must also be provided when checking. For example:

```
$ HTTPD /DO=AUTH=CHECK /SYSUAF /PROFILE
```

A server's currently loaded configuration can be interrogated from the Server Administration menu (see "WASD Web Services - Features and Facilities").

4.1 Include File Directive

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

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

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

```
# WASD_CONFIG_MAP

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

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

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

Note

Such configurations cannot be managed using Server Administration facility (see

"WASD Web Services - Features and Facilities"). Files containing [IncludeFile] directives are noted during server startup and if an Server Administration page configuration interface is accessed where this would be a problem an explanatory message and warning is provided. A configuration *can still be saved* but the resulting configuration will be a flat-file representation of the server configuration, not the original hierarchical one.

4.2 Site Organisation

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 WASD_ROOT:[000000], the latter perhaps in something like WEB:[000000]. This logical device could be provided with the following DCL introduced into the site or server startup procedures:

```
$ DEFINE /SYSTEM /TRANSLATION=CONCEALED WEB DKA0:[WEB.]
```

See Section 12.2 for further information on the use of logical names in locating and defining the content and structure of a site.

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

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

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

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

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

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

There are no good reasons for having site-specific documents integrated into the package directory structure!

All site-served files should be located in an autonomous, dedicated area or areas. The only reason to place script files into WASD_ROOT:[CGI-BIN] or WASD_ROOT:[*architecture*_BIN] is that the script script is traditionally accessible via a /cgi-bin/ path or that the site is a small and/or low usage environment where this directory is conveniently available for the few extra scripts being made available.

For any significant site (size that as best suits your perception), or for when a specific software system or systems is being built or exists and it is being "Web-ified", design that software system as you would be any other. That is place the documentation in one directory are, executables and support procedures in their own, management files in another, data in yet another area, etc. Then make those portions that are required to be accessible via the Web interface accessible via the logical associations afforded through the use of the server's mapping rules (Chapter 12). Of course existing areas that are to be now made available via the Web can be mapped in the same way. This includes the active components - executable scripts. There is no reason (apart from historical) why the /cgi-bin/ path should be used to activate scripts associated with a dedicated software system. Use a specific and unique path for scripts associated with each such system.

When making a directory structure available via the Web care must be taken that only the portions required to be accessed can be. Other areas should or must not be accessible. The server process can only access files that are world-accessible, it is specifically granted access via VMS protection mechanisms (e.g. ACLs), or that the individual SYSUAF-authorized accessor can access and which have specifically been made available via server authorization rules. Use the recommendations in Section 5.2 as guidlines when designing your own site's protections and permissions.

Document Root

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

For example, the following WASD_CONFIG_MAP rule set

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

pass /*/-/* /wasd_root/runtime/*/*
/wasd_root/* /wasd_root/*

set * map=root=/dka0/the_site

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

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

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

access to the document represented by file

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

With the request for a directory icon using

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

access to the image represented by file

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

And a request for a script using

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

activation of the script represented by the file

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

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

4.3 Virtual Services

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

WASD_CONFIG_SERVICE

If the logical name WASD_CONFIG_SERVICE is defined the deprecated WASD_CONFIG_GLOBAL [Service] directive is not used (see below).

See Chapter 9 for further detail.

WASD_CONFIG_GLOBAL [Service] *(Deprecated)*

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

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

```
[Service]
alpha.example.com
alpha.example.com:8080
beta.example.com
beta.example.com:8000
```

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

4.3.1 [[virtual-server]]

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

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

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

Service Conditionals

As a virtual service specification acts as a conditional on subsequent rule application they must be considered a fundamental element of Chapter 7. Service conditionals also impose a boundary on the scope of *if..endif* constructs.

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

Note

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

to provide it with mapping rules, etc. The server will complain in instances where it is redundant.

4.3.2 Unknown Virtual Server

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

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

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

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

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

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

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

4.4 GZIP Encoding

WASD can apply GZIP compression (gzip, deflate) to any suitable response body and can accept similarly compressed request bodies. It dynamically maps required functions from a ZLIB shareable image. Originally developed against the ZLIB v1.2.*n* port by Jean-François Piéronne, the VMS-PORTS (GNV) LIBZ package is also supported.

WASD dynamically maps the associated shareable image by successively accessing the (optionally defined) WASD_LIBZ_SHR32 logical name, then GNV$LIBZSHR32, then LIBZ_SHR32, before reporting GZIP unavailable.

The shareable image must be INSTALLed (without any particular privileges) before it can be activated by the privileged WASD HTTPd image (the WASD startup will automatically do this if necessary). The server process log and the Server Administration page, Statistics Report panel named Environment contains the version activated or a VMS status message if an error was encountered.

4.4.1 Response Encoding

The WASD_CONFIG_GLOBAL directive [GzipResponse] controls whether this feature is enabled for the gzip content-encoding of suitable response bodies. This directive requires at least one parameter, the compression level in the range 1..9. Smaller values provide faster but poorer compression ratios while larger values better compression at the cost of more CPU cycles and latency. This corresponds to the GZIP utility's -1..-9 CLI switches. Two optional parameters could allow ZLIB's 'memLevel' and 'windowBits' to be adjusted by ZLIB afficiendos (level[,memory,window]). A small amount of experimentation by this author indicates minor changes in memory usage and compression ratio by fiddling with these.

Be aware that GZIP encoding is **memory intensive**. From 132kB to 265kB has been observed per compressing request (WATCH provides this in a summary line). These values apply across a wide range of transfer sizes (from kilobytes to tens of megabytes). It also is **CPU intensive** and adds response latency, though that might be well be offset by significant reductions in transfer time on the Internet or other slower, non-intranet infrastructures. Text content compression has been observed from 30% to 10% of the original file size (even down to 1% in the case of the extremely redundant content of [EXAMPLE]64K.TXT). VMS executables (for want of another binary test case) at around 40%. In other words, GZIP encoding may not be suitable or efficient for every site or every request!

Once enabled WASD will GZIP the responses for all suitable contents provided the client accepts the encoding and the response is not one of the following:

- less than 1400 bytes (no point in the overhead)
- already content-encoded script output
- a compressed image (e.g. GIF, JPEG, PNG, etc)
- a video stream (presumably already compressed, e.g. MPEG)
- a compressed audio stream
- a PDF file
- a Shockwave Flash file
- an obviously compressed application stream (e.g. GZIP, ZIP, JAR)

Additional control may be exercised with the following path SETings:

- "response=GZIP=all", matching paths will always have GZIP encoding performed (the above constraints still apply)
- "response=GZIP=none", matching paths will never have GZIP encoding
- "response=GZIP=<integer>", responses with content-lengths greater than the specified number of kilobytes will be GZIP content-encoded (if the content-length cannot be determined it will NOT not encoded and the above constraints still apply)

Using path settings GZIP compression may be disabled for specified file types (apart from those already suppressed as described above).

```
set **.myzip response=gzip=none
```

A script using the *Script-Control: X-content-encoding-gzip=0* CGI response header can similarly suppress GZIP compression of its output if required. See “Scripting Overview” for further detail.

Flush Period

By default GZIP encoding flushes the internal buffer only when full. Most commonly this is not an issue because of high rates of output. However with slow output sources, such as from some classes of script, this can result in considerable latency before a client sees an initial response, and then between transmission of further output. By default output is initially flushed after 5 seconds and thereafter at a maximum interval of 15 seconds. The WASD_CONFIG_GLOBAL directive [GzipFlushSeconds] allows this period to be adjusted.

4.4.2 Request Encoding

Decoding of GZIP content-encoded request bodies is enabled using the WASD_CONFIG_GLOBAL directive [GzipAccept]. Enabling this using a value 15 (or 1) results in the server advertising its acceptance of GZIPed requests using the "Accept-Encoding: gzip, deflate" response header. Requests containing bodies GZIP compressed will have these decoded as they are read from the client and before further processing, such as the upload of files into server accessible file-system space. This decoding is optional and not the default with DCL and DECnet script processing. That is, a request body will be passed to the script still encoded unless specific mapping directs otherwise. Decoding by the server into the original data prior to transfering to the script can be enabled for all or selected scripts using the following path settings:

- “script=body=decode”, script gets the decoded stream
- “script=body=NOdecode”, script gets the raw, encoded stream (default)

Note that scripts need to be specially aware of both GZIP encoded bodies and those already decoded by the server. In the first case the stream must be read to the specified content-length and then decoded. In the second case, a content-length cannot be provided by the server (without unencoding the entire stream ahead of time it cannot predict the final size). Where the server is to decode the request body before transfering it to the script it changes the CGI variable CONTENT_LENGTH to a single question-mark ("?"). Scripts may use this to detect the server’s intention and then must ignore any transfer-encoding and/or content-encoding header information and read the request body until end-of-file is received.

GZIP decoding (decompression) is understandably much less memory and CPU intensive. Experimentation indicates it does not contribute significantly to latency either.

4.5 Request Throttling

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

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

This is applied to a path (or paths) using the WASD_CONFIG_MAP mapping SET THROTTLE= rule (Section 12.5.5). The general format is

```
set path throttle=n1[/u1][,n2,n3,n4,t/o1,t/o2]
set path throttle=from[/per-user][,to,resume,busy,t/o-queue,t/o-busy]
```

where

- *n1* sets the number of concurrent requests before queuing begins (the number of processing requests becomes static and the number of queued requests increases)
- *u1* is separated from the *n1* value by a forward-slash and limits the concurrent request any one authenticated user can process. Even though the *n1* value may allow processing if *u1* would be exceeded the request is queued.
- *n2* is the concurrent requests before FIFO queuing begins, meaning each new request is put onto the queue but at the same the first-in request is taken off the queue for processing (the number of queued requests becomes static and the number of processing requests increases)
- *n3* puts a limit on FIFO queuing (the number of queued requests again increases and the number of processing requests becomes static)
- *n4* is an absolute limit for concurrent requests against the path (a 503 "server too busy" status is immediately generated)
- *t/o1* is the maximum period for queued requests before they are processed (if not constrained by *n3*)
- *t/o2* is the maximum period for queued requests before a 503 "server too busy" response is returned, it begins immediately or following the expiry of any *t/o1*

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

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

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

A comprehensive throttle statistics report is available from the Server Administration facility.

Per-User Throttle

If the concurrent processing value (*n1*) has a second, slash-delimited integer, this serves to limit the number of authenticated user-associated requests that can be concurrently processing.

When a request is available for processing the associated remote user name is checked for activity against the queue. The *u1* (or per-user throttle value) is a limit on that user name's concurrent processing. If it would exceed the specified value the request is queued until the number of requests processing drops below the *u1* value. All other values in the throttle rule are applied as for non-per-user throttling.

Note

The user name used for comparison purposes is the authenticated remote user (same as the CGI variable value REMOTE_USER). This can be for any realm. Of course the same string can be used to represent different users within different authentication realms and so care should be exercised that per-user throttling does not span realms otherwise unexpected (and incorrect) throttling may occur for distinct users.

If an unauthenticated request is matched against the throttle rule (i.e. there is no authorization rule matching the request path) the client has a 500 (server error) response returned. Obviously per-user throttling must have a remote user name to throttle against and this is a configuration issue.

Examples

1. **throttle=10**

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

2. **throttle=10,20**

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

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

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

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

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

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

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

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

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

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

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

8. **throttle=10/1**

   Concurrent requests up to 10 are processed. The requests must be of authenticated users. Each authenticated user is allowed to execute at most one concurrent request against this path. When 10 is reached, or if less than 10 users are currently executing requests, then further requests are queued to server capacity.

9. **throttle=10/1,,,,,00:03:00**

   This is basically the same as scenario 8) but with a busy-on-timeout of three minutes. When the timeout expires any requests still queued against the user name is immediately dequeued with a 503 "busy" status.

Mapping Reload

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

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

4.6 Client Concurrency

The “client_connect_gt:” mapping conditional (Chapter 7) attempts to allow some measurement of the number of requests a particular client currently has being processed. Using this decision criterion appropriate request mapping for controlling the additional requests can be undertaken. It is not intended to provide fine-grained control over activities, rather just to prevent a single client using an unreasonable proportion of the resources.

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

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

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

4.7 Content-Type Configuration

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

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

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

4.7.1 Adding Content-Types

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

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

Note

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

Mappings using [AddType] look like these.

```
[AddType]
.html  text/html   Web 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
```

4.7.2 MIME.TYPES

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

Mappings using MIME.TYPES look like these.

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

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

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

- **#! file description**
  A space reserved character indicates following free-form text, used as the file type description displayed on the far right of directory listings.
- **#!/cgi-bin/script**
  A forward-slash introduces an auto-script specification. An auto-script is automatically activated by the server to process and display a corresponding file's contents. These are sometimes refered to as *presentation* scripts.
- **#![*alt*] /path/to/icon.gif**
  A left-square-bracket is used for icon specifications. These are actually mapped against the following content-type, not file suffix, and so only need to be specified once for each content-type in the file. This behaves in a similar fashion to [AddIcon], only the components are reversed.
- **#!!**
  The two exclamation marks can be used to indicate a MIME type intended for WASD only. The can be ignored by non-WASD applications.

- **#!+**
  An exclamation mark then a plus symbol indicates an FTP transfer mode directive. One of three characters may follow the plus. An “A” indicates that this file type should be FTP transfered in ASCII mode. An “I” or a “B” indicates that this file type should be FTP transfered in Image (binary) mode.
- **#!%**
  A percentage is ignored by WASD. This is reserved for local (non-WASD) viewers.

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

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

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

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

4.7.3 Unknown Content-Types

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

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

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

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

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

4.7.4 Explicitly Specifying Content-Type

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

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

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

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

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

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

This is an example:

online demonstration

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

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

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

This is an example:

online demonstration

4.8 Language Variants

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

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

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

or

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

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

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

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

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

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

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

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

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

Content-Type

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

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

Non-Text Content

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

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

4.9 Character Set Conversion

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

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

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

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

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

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

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

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

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

4.10 Error Reporting

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

4.10.1 Basic and Detailed

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

```
ERROR 404 - The requested resource could not be found.
Document not found ... /wasd_root/index.html
(document, bookmark, or reference requires revision)
Additional information: 1xx, 2xx, 3xx, 4xx, 5xx, Help
---------------------------------------------------------------
WASD/10.0.0 Server at www.example.com Port 80
```

There is also the more basic.

```
ERROR 404 - The requested resource could not be found.
Additional information: 1xx, 2xx, 3xx, 4xx, 5xx, Help
---------------------------------------------------------------
WASD/10.0.0 Server at www.example.com Port 80
```

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

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

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

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

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

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

Other Customization

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

```
ERROR 404 - The requested resource could not be found.
---------------------------------------------------------------
WASD/10.0.0 Server at www.example.com Port 80
```

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

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

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

4.10.2 Site Specific

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

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

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

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

To exclude selected error reports (and handle all others by default) add the HTTP status codes preceded by a hyphen following the report path. In this example 401 and 500 errors are server-generated.

```
[ErrorReportPath] /httpd/-/reporterror.shtml -401 -500
```

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

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

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

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

```
# WASD_CONFIG_GLOBAL
[ErrorReportPath] /errorreport.shtml

# WASD_CONFIG_MAP
[[alpha.example.com]]
pass /errorreport.shtml /httpd/-/alphareport.shtml
[[beta.example.com]]
pass /errorreport.shtml /httpd/-/betareport.shtml
[[gamma.example.com]]
pass /errorreport.shtml /httpd/-/gammareport.shtml
```

Using Static HTML Documents

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

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

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

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

Using an SSI Document

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

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

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

Error Variables

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

Using a Script

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

4.11 OPCOM Logging

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

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

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

4.12 Access Logging

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

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

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

Exclude requests from specified hosts using the [LogExcludeHosts] configuration parameter, or using the "SET NOLOG" mapping directive.

4.12.1 Log Format

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

- **COMMON -** This is the most common, base logging format for Web servers. COMMON is the default log format.
- **COMMON_SERVER -** This is an optional format used, for one, by the NCSA server. It is basically the common format, with the server host name appended to the line (used for multi-homed servers, see Section 4.3).

- **COMBINED -** This is an optional format used, for one again, by the NCSA server. It too is basically the common format, with the HTTP referer and user agent appended.

User-Defined

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

Two different "escape" characters introduce the following parameters:

A "!" followed by

Characters	Description
AR	authentication realm (if any)
AU	authenticated user name (if any)
BB	bytes in body (excludes response header)
BQ	quadword bytes in response (includes header)
BY	bytes in response (includes header)
CA	client address
CC	X509 client certificate authorization distinguishing name
CI	SSL session cipher (e.g. "AES128-SHA", "AES256-SHA256")
CL	value provided by "Content-Length:" header (cf. "PL")
CN	client host name (or address if DNS lookup disabled)
CP	client port
DI	specified dictionary value
ID	session track ID - obsolete
EM	request elapsed time in milliseconds
ES	request elapsed time in fractional seconds
ME	request method
NP	specified notepad value
PA	request path (not to be confused with "RQ")
PL	actual body (payload) length received with POST or PUT (cf. "CL")
PR	request URL (includes protocol scheme)
QS	request query string (if any)

Characters	Description
RF	referer (if any)
RQ	complete request string (see below)
RP	request protocol
RS	response status code
SN	server host name
SC	script name (if any)
SM	request scheme (http: or https:)
SP	server port
SR	SSL session reused
SV	SSL protocol (e.g. “SSLv3”, “TLSv1”)
TC	request time (common log format)
TI	request time (local in ISO 8601 extended format)
TS	request time (UTC in ISO 8601 basic format) sortable
TU	request time (UTC)
TV	request time (VMS format)
UA	user agent
VS	virtual service (service host:port)
XX	custom, usually site/client-specific, logging item see module [SRC.HTTPD]LOGGING.C functions LoggingCustom..( )

A “\” followed by

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

Any other character is directly inserted into the log entry.

“PA” and “RQ”

The “PA” and “RQ” have distinct roles. In general the “RQ” (request) directive will always be used as this is the full request string; script component (if any), path string

and query string component (if any). The “PA” directive is merely the path string after any script and query string components have been removed.

Pre-defined Plus User-Defined

It is possible to use one of the pre-defined log format keywords with additional user-defined directive appended. The appended directives must include ALL additional literal characters and directives required in the log entry. The syntax is <pre-defined keyword>+<appended format> as in “COMMON+ !EM”.

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
```

5. Alternatively, to append the SSL protocol version and cipher with the combined format:

```
COMBINED+ !SV !CI
```

4.12.2 Log Per-Period

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

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

The log file changes on the first request after the entering of the new period.

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

- server host name
- server port
- year (YYYY)
- month (MM)
- day (DD)
- hour (HH, only present when HOURLY period is configured)

as in the following example

```
WASD_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.

4.12.3 Log Per-Service

By default a single access log file is created for each HTTP server process. Using the [LogPerService] configuration directive a log file for each service provided by the HTTPd is generated (Section 4.3). The [LogNaming] format can be any of "NAME" (default) which names the log file using the first period-delimited component of the IP host name, "HOST" (which uses as much of the IP host name as can be accomodated within the maximum 39 character filename limitation under ODS-2), or "ADDRESS" which uses the full IP host address in the name. Both HOST and ADDRESS have hyphens substituted for periods in the string. If these are specified then by default the service port follows the host name component. This may be suppressed using the [LogPerServiceHostOnly] directive, allowing a minimum extra 3 characters in the name, and combining entries for all ports associated with the host name (for example, a standard HTTP service on port 80 and an SSL service on port 443 would have entries in the one file).

4.12.4 Log Per-Instance

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

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

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

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

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

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

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

4.12.5 Log Naming

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

- ADDRESS
- HOST
- NAME (default)

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

```
WASD_LOGS:131-185-250-202_80_ACCESS.LOG
WASD_LOGS:WWW-EXAMPLE-COM_80_ACCESS.LOG
WASD_LOGS:WASD_80_ACCESS.LOG
```

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

```
WASD_LOGS:131-185-250-202_80_19971013_ACCESS.LOG
WASD_LOGS:WWW-EXAMPLE-COM_80_19971013_ACCESS.LOG
WASD_LOGS:WWW_80_19971013_ACCESS.LOG
```

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

```
WASD_LOGS:131-185-250-202_80_ACCESS.LOG_KLAATU_HTTPD-80
WASD_LOGS:WWW-EXAMPLE-COM_80_ACCESS.LOG_KLAATU_HTTPD-80
WASD_LOGS:WASD_80_ACCESS.LOG_KLAATU_HTTPD-80
WASD_LOGS:131-185-250-202_80_19971013_ACCESS.LOG_KLAATU_HTTPD-80
WASD_LOGS:WWW-EXAMPLE-COM_80_19971013_ACCESS.LOG_KLAATU_HTTPD-80
WASD_LOGS:WWW_80_19971013_ACCESS.LOG_KLAATU_HTTPD-80
```

4.12.6 Access Tracking

Access tracking has been obsoleted with WASD v11.0.

4.12.7 Access Alert

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

ALERT - at the conclusion of the request
ALERT=MAP - immediately after mapping (early)
ALERT=AUTH - when (any) authorization has been performed
ALERT=END - at the conclusion of the request (default)
ALERT=*integer* - see below
NOALERT - suppress alert for this path

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

Chapter 5
Security Considerations

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

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

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

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

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

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

- WASD_ROOT:[DOC.MISC]WASD_ADVISORY_020925.TXT
- https://www.cvedetails.com/cve/CVE-2002-1825/

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

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

5.1 Server and Site Testing

This is the merest of mentions for a topic that literally encompasses volumes!

Each site is very-much an individual combination of configurations and applications. Each site therefore has specific potential vulnerabilities that should be known about and addressed where possible. Especially if you have an Internet-facing site then **this mean you!**

Many tools exist at the time of writing that didn't fifteen years before when WASD was investigated as described above. Some are on-line, "free" site health checks and penetration testing. Others are tools that can (often) be used from your platform of choice, many of which are free and open-source (FOSS). We are spoiled for choice.

In WASD's earlier years tools such as *Apache Bench*, *WASD Bench*, along with batched *cURL* and *wget* requests were used to exercise and, in some limited fashion, *fuzz* the server (providing invalid, unexpected, or random request data) in an effort to discover flaws in server code and execution.

Currently the WASD development bench uses the OWASP ZAP tool to provide a much more comprehensive exercise and test environment.

OWASP ZAP

"Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project (OWASP). ZAP is designed specifically for testing web applications and is both flexible and extensible.

. . .

ZAP provides functionality for a range of skill levels from developers, to testers new to security testing, to security testing specialists. ZAP has versions for each major OS and Docker, so you are not tied to a single OS. Additional functionality is freely available from a variety of add-ons in the ZAP Marketplace, accessible from within the ZAP client."

https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP is cross-platform (Linux, macOS, Windows, other), GUI-based, Java-implemented, and may be used effectively, though certainly not to its full capabilities, after fifteen minutes with the introductory documentation. **ZAP is a highly recommended tool for site vulnerability assessment.**

ZAP is used to exercise the in-development WASD, in particular the following aspects (not in any particular order).

- **Traffic Loading -** server behaviour under load; continuing to process correctly while not exhibiting bottlenecks in performance, or worse, failing with soft (internal assertion checking) or hard (e.g. ACCVIO) bugchecks. Latency in AST-based processing often reveals subtle dependencies, race conditions, or other timing-related issues. ZAP allows a configurable number of concurrent requests when both spidering and vulnerability scanning.
- **Graded Alerts -** reports and counts of known attack vectors or general recommendations after spidering or penetration scans. These are flagged as high, medium or low risk, provide descriptions with references, and a quick overview of mitigation strategies. Each instance encountered during the scan has the request-response data available for analysis allowing specific cases to be identified and mitigated.
- **Directory Traversal -** (also known as path traversal) aims to access files and directories that are stored outside the server root, web root or web application folders. By manipulating data that reference files with *dot-dot-slash* (../) sequences and its variations, or by using absolute file paths, it may be possible to access arbitrary files and directories stored in the server or general file system.
- **Data Injection -** covers a variety of attacks where request parameters are used to execute (CLI) commands, SQL queries, interpreted script code (e.g. JavaScript, PHP), or platform-executable binary code. Injecting encoded or obscured data into an HTTP request via the query-string or header field values is a common vector. Lack of appropriate data validation underlies injection vulnerability.
- **Buffer Overflow -** the overwriting of memory fragments of the process, which should never be modified intentionally or unintentionally. HTTP requests with unusually large or otherwise unintended header field values, or web application input fields designed for small, fixed-length, or specific type data are obvious targets. Fuzzing requests can often induce this.
- **Request Fuzzing -** where malformed or spurious data is automatically generated and injected into the processing in an effort to induce unexpected behaviour or failure. In web environments this can include the HTTP protocol itself, the specific implementation of some capability of the server, and any scripting environment or web application hosted on a server.
- **Cross Site Scripting -** where a malicious web element such as JavaScript, HTML, or other browser-side code is injected into otherwise benign and trusted web content from a non-same-origin, third-party source.

It should be noted that these are provided “out-of-the-box”, is a subset of that *out-of-the-box* functionality of particular interest in WASD development, and utilise only a tiny percentage of ZAP total capabilities.

5.2 Recommended Package Security

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

Some pre-v8.1 directories are not included in this table. These are not significant in versions from 8.1 onwards and may be deleted. They can continue to exist however and the security procedures described below ensure that they comply to the general post-8.1 security model. The file access permissions indicated below are for directory contents. The directory files themselves have settings appropriate for content access.

Package Access

Directory	Access World	Access Other	Description
[AXP-BIN]	none	script:RE	Alpha executable script files
[AXP]	none	none	Alpha build and utility area
[CGI-BIN]	none	script:RE	architecture-neutral script files
[DOC]	read	(world)	package documentation
[EXAMPLE]	read	(world)	package examples
[EXERCISE]	read	(world)	package test files
[HTTP$NOBODY]	none	script:RWED	scripting account default home area
[HTTP$SERVER]	none	server:RWED	server account default home area
[IA64-BIN]	none	script:RE	Itanium executable script files
[IA64]	none	none	Itanium build and utility area
[INSTALL]	read	(world)	installation, update and secuity procedures
[LOCAL]	none	none	site configuration files
[LOG]	none	none	site access logs
[LOG_SERVER]	none	server:RWED	server process (SYS$OUTPUT) logs
[RUNTIME]	read	(world)	graphics, help files, etc.
[SCRATCH]	none	script:RWED	working file space for scripts
[SCRIPT]	none	none	example architecture-neutral scripts

Directory	Access World	Access Other	Description
[SRC]	none	(world)	package source files
[STARTUP]	none	server:RE	package startup procedures
[VAX-BIN]	none	script:RE	VAX executable script files
[VAX]	none	none	VAX build and utility area

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

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

Rights Identifiers

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

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

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

Directory WASD_ROOT:[000000]

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

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

Directory WASD_ROOT:[CGI-BIN]

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

Total of 1 file.
```

5.3 Maintaining Package Security

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

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

SECURE.COM

The following DCL procedure resets the package security baseline.

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

It guides the administrator through a number of stages

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

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

SECHAN Utility

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.4 Independent Package and Local Resources

Not only does it make it easier to manage site content but is also good security practice to keep server package and site content completely separate (Section 4.2).

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

5.5 Configuration

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

5.5.1 Directory Listings

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

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

The mapping rule "SET DIR=*keyword*" can be used to change this on a per-path basis (Section 12.5.5).

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

5.5.2 Server Reports

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

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

The mapping rule "SET REPORT=*keyword*" can be used to change some of these on a per-path basis (Section 12.5.5).

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

5.5.3 Scripting

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

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

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

Also see Section 5.6.

5.5.4 Server Side Includes

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

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

The mapping rule "SET SSI=*keyword*" can be used to change some of this on a per-path basis (Section 12.5.5).

Conservative recommendation: Disable [SsiExec].

5.6 Scripting

Scripting has been a notorious source of server compromise, particularly within Unix environments where script process shell command-line issues require special attention. The WASD CGI scripting interface does not pass any arguments on the command line, and is careful not to allow substitution when constructing the CGI environment. Nevertheless, script behaviours cannot be guaranteed and care should be exercised in their deployment (ask me!)

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

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

Default Accounts

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

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

5.7 Authorization

Authorization issues imply controlling access to various resources and actions and therefore require careful planning and implementation if compromise is to be avoided. WASD has a quite capable and versatile authorization and authentication environment, with a significant number of considerations.

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

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

5.8 Miscellaneous Issues

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

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

Package Tree

Version 8.1 and later is much more conservative in what it makes available of the package tree via the server. The package installation, update and security procedures and their associated utilities should always be used to ensure that the installed package continues to conform to the security baseline. See Section 5.3.

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

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

```
# WASD_CONFIG_MAP
pass /wasd_root/doc/*
pass /wasd_root/src/*
pass /wasd_root/example/*
pass /wasd_root/exercise/*
fail /wasd_root/*
```

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

```
# WASD_CONFIG_MAP
pass /wasd_root/*

# WASD_CONFIG_AUTH
[WASD_WEB_ADMIN=id]
/httpd/-/admin/* r+w
/wasd_root/local/* r+w
/wasd_root/* r
```

Be careful!

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

The following example shows how this might occur.

```
# WASD_CONFIG_MAP
fail /wasd_root/doc/*
pass /* /wasd_root/*
```

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

```
# WASD_CONFIG_AUTH
if (path-translated:WASD_ROOT:[DOC]*) * none
```

or to selectively allow access

```
# WASD_CONFIG_AUTH
[[WASD_VMS_RW=id]]
if (path-translated:WASD_ROOT:[DOC]*) * read
```

5.9 Site Attacks

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

Platform Vulnerabilities

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

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

Site Vulnerabilities

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

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

Denial of Service

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

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

Password Cracking

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

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

Authorization Holes

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

WASD's /AUTHORIZATION=ALL functionality may assist here (Section 5.7).

Strategies

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

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

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

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

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

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

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

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

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

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

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

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

Chapter 6
String Matching

Matching of strings is a pervasive and important function within the server. Two types are supported; wildcard and regular expression. Wildcard matching is generally much less expensive (in CPU cycles and time) than regular expression matching and so should always be used unless the match explicitly requires otherwise. WASD attempts to improve the efficiency of both by performing a preliminary pass to make simple matches and eliminate obvious mismatches using a very low-cost comparison. This either matches or doesn't, or encounters a pattern matching meta-character which causes it to undertake full pattern matching.

To assist with the refinement of string matching patterns the Server Administration facility has a report item named "Match". This report allows the input of target and match strings and allows direct access to the server's wildcard and regular expression matching routines. Successful matches show the matching elements and a substitution field (Section 6.4) allows resultant strings to be assessed.

To determine what string match processing is occuring during request processing in the running server use the *match* item available from the Server Administration WATCH Report.

6.1 Wildcard Patterns

Wildcard patterns are simple, low-cost mechanisms for matching a string to a template. They are designed to be used in path and authorization mapping to compare a request path to the root (left-hand side) or a template expression.

Wildcard Operators

Expression	Purpose
*	Match zero or more characters (non-greedy)
**	Match zero or more characters (greedy)
%	Match any one character

Wildcard matching uses the '*' and '%' symbols to match any zero or more, or any one character respectively. The '*' wildcard can either be greedy or non-greedy depending on the context (and for historical reasons). It can also be forced to be greedy by using two consecutive ('**'). By default it is not greedy when matching request paths for mapping or authentication, and is greedy at other times (matching strings within conditional testing, etc.)

Greedy and Non-Greedy

Non-greedy matching attempts to match an asterisk wildcard up until the first character that is not the same as the character immediately following the wildcard. It matches a minimum number of characters before failing. Greedy matching attempts to match all characters up until the first string that does not match what follows the asterisk.

To illustrate; using the following string

```
non-greedy character matching compared to greedy character matching
```

the following non-greedy pattern

```
*non-greedy character*matching
```

does not match but the following greedy pattern

```
*non-greedy character**matching
```

does match. The non-greedy one failed as soon as it encountered the space following the first "matching" string, while the greedy pattern continued to match eventually encountering a string matching the string following the greedy wildcard.

6.2 Regular Expressions

Regular expression matching is case insensitive (in line with other WASD behaviour) and uses the POSIX EGREP pattern syntax and capabilities. Regular expression matching offers significant but relatively expensive functionality. One of those expenses is expression compilation. WASD attempts to eliminate this by pre-compiling expressions during server startup whenever feasable. Regular expression matching must be enabled using the [RegEx] WASD_CONFIG_GLOBAL directive and are then differentiated from wildcard patterns by using a leading "^" character.

A detailed tutorial on regular expression capabilities and usage is well beyond the scope of this document. Many such hard-copy and on-line documents are available.

```
http://en.wikipedia.org/wiki/Regular_expression
```

This summary is only to serve as a quick mnemonic. WASD regular expressions support the following set of operators.

Operator Overview

Description	Usage
Match-self Operator	Ordinary characters.

Description	Usage
Match-any-character Operator	.
Concatenation Operator	Juxtaposition.
Repetition Operators	* + ? {}
Alternation Operator	|
List Operators	[...] [^...]
Grouping Operators	(...)
Back-reference Operator	\digit
Anchoring Operators	^ $
Backslash Operator	Escape meta-character; i.e. \ ^ . $ | [(

The following operators are used to match one, or in conjunction with the repetition operators more, characters of the target string. These single and leading characters are reserved meta-characters and must be escaped using a leading backslash ("\") if required as a literal character in the matching pattern. **Note** that this does not apply to the *range* hyphen; to include a hyphen in a range ensure the character is the first or last in the range.

Matching Operators

Expression	Purpose
^	Match the beginning of the line
.	Match any character
$	Match the end of the line
|	Alternation (or)
[abc]	Match only a, b or c
[^abc]	Match anything except a, b and c
[a-z0-9]	Match any character in the range a to z or 0 to 9

Repetition operators control the extent, or number, of whatever the matching operators match. These are also reserved meta-characters and must be escaped using a leading backslash if required as a literal character.

Repetition Operators

Expression	Function
*	Match 0 or more times

Expression	Function
+	Match 1 or more times
?	Match 1 or zero times
{n}	Match exactly n times
{n,}	Match at least n times
{n,m}	Match at least n but not more than m times

6.3 Examples

The following provides a series of examples as they might occur in use for server configuration.

1. Equivalent functionality using wildcard and regular expression patterns. Note that "Mozilla" must be at the start of the string, with the regular expression using the start-of-string anchor resulting in two consecutive "^"s, one indicating to WASD a regular expression, the other being part of the expression itself.

```
if (user-agent:Mozilla*Gecko*)
if (user-agent:^^Mozilla.*Gecko)
```

2. This shows path matching using equivalent wildcard and regular expression matching. Note the requirement to use the regular expression *grouping* parentheses to provide the substitution elements, something provided implicitly with wildcard matching.

```
map /*/-/* /wasd_root/runtime/*/*
map ^/(.+)/-/(.+) /wasd_root/runtime/*/*
```

3. This rather contrived regular expression example has no equivalent capability available with wildcard matching. It forbids the use of any path that contains any character other than alpha-numerics, the hyphen, underscore, period and forward-slash.

```
pass ^[^-_./a-z0-9]+ "403 Forbidden character in path!"
```

6.4 Expression Substitution

Expression substitution is available during path mapping (Chapter 12). Both wildcard (implicitly) and regular expressions (using *grouping* operators) note the offsets of matched portions of the strings. These are then used for wildcard and *specified* wildcard substitution where result strings provide for this (e.g. mapping 'pass' and 'redirect' rules). A maximum of nine such wildcard substitutions are supported (one other, the zeroeth, is the full match).

Wildcard Substitution

With wildcard matching each asterisk wildcard contained in the pattern (*template* string) has matching characters in the *target* string noted and stored. Note that for the percentage (single character) wildcard no such storage is provided. These characters are available for substitution using corresponding wildcards present in the *result* string. For instance, the target string

```
this is an example target string
```

would be matched by the pattern string

```
* is an example target *
```

as containing two matching wildcard strings

```
this
string
```

which could be substituted using the result string

```
* is an example result *
```

producing the resultant string

```
this is an example result string
```

Regular Expression Substitution

With regular expression matching the groups of matching characters must be explicitly specified using the *grouping* parenthesis operator. Hence with regular expression matching it is possible to match many characters from the target string without retaining them for later substitution. Only if that match is designated as a subsitution source do the matching characters become available for substituion via any result string. Using two possible target strings as an example

```
this is an example target string
this is a contrived target string
```

would both be matched by the regular expression

```
^^([a-z]*) is [a-z ]* target ([a-z]*)$
```

which though it contains three regular expressions in the pattern, only two have the grouping parentheses, and so make their matching string available for substitution

```
this
string
```

which could be substituted using the result string

```
* is the final result *
```

producing the resultant string

```
this is the final result string
```

Specified Substitution

By default the strings matched by wildcard or grouping operators are substituted in the same order in which they are matched. This order may be changed by specifying which wildcard string should be substituted where. Not all matched (and stored) strings need to be substituted. Some may be omitted and the contents effectively ignored.

The specified substitution syntax is a result wildcard followed by a single-apostrophe (') and a single digit from zero to nine (0 . . . 9). The zeroeth element is the full matching string. Element one is the first matching part of the expression, on through to the last. Specifying an element that had no matching string substitutes an empty string (i.e. nothing is added). Using the same target string as in the previous previous example

```
this is an example target string
```

and matched by the wildcard pattern string

```
* is an example target *
```

when substituted by the result string

```
*'2 is an example result
```

would produce the resultant string

```
string is an example result
```

with the string represented by the first wildcard effectively being discarded.

Chapter 7
Conditional Configuration

Request processing (WASD_CONFIG_MAP) and authorization (WASD_CONFIG_AUTH) rules may be conditionally applied depending on request, server or other charactersistics. These include

server host name, port
client IP address and host name
browser-accepted content-types, character sets, languages, encodings
browser identification string
scheme ("http:" or "https:", i.e. is it a secure request?)
HTTP method (GET, POST, etc.)
request path, query string, cookie data, refering page
virtual host:port specified in request header
system information (hardware, Alpha/VAX, node name, VMS version, etc.)
local time
random number generation

7.1 Service Conditionals

As described in Section 4.3.1 a [[*host*:*port*]] rule applies subsequent configuration depending on whether the request service matches the specified service. This makes it a fundamental element of conditional configuration.

Note that service conditionals impose a boundary on the scope of *if..endif* constructs. That is, an *if..endif* may not span a virtual service conditional. A conditional flow syntax error is reported if an *if..endif* construct is not properly closed before encountering a subsequent [[*host*:*port*]] rule.

7.2 If..endif Conditionals

These may be nested up to a maximum depth of eight, are not case sensitive and generally match via string comparison, although some tests are performed as boolean operations, by converting the conditional parameter to a number before comparison, and IP address parameters will accept a network mask as well as a string pattern.

String Matching

The basis of much conditional decision making is string pattern matching. Both wildcard and regular expression based pattern matching is available (Chapter 6). Wildcard matching in conditional tests is *greedy*. Regular expression matching, in common with usage throughout WASD, is differentiated from wildcard patterns using a leading "^" character.

Conditional Syntax

Conditional expressions and processing flow structures may be used in the following formats. Conditional and rule text may be indented for clarifying structure.

```
if (condition) then apply rest of line
```

```
if (condition)
   then apply one
   or more rules
   up until the corresponding  . . .
endif
```

```
if (condition)
   then apply one
   or more rules
else
   apply one or more other rules
   up until the corresponding  . . .
endif
```

```
if (condition)
   then apply one
   or more rules
elif (condition)
   apply one or more other rules
   in a sort or case statement
else
   a possible default rule or rules
   up until the delimiting
endif
```

Logical operators are also supported, in conjunction with precedence ordering parentheses, allowing moderately complex compound expressions to be applied in conditionals.

! logical negation
&& logical AND
| | logical OR

There are two more conditional structures that allow previous decisions to be reused. These are *unif* and the *ifif*. The first unconditionally includes rules regardless of the current state of execution. The second resumes execution only if the previous *if* or *elif* expression was true. The *else* statement may also be used after an *unif* to continue only if the previous expression was false. The purpose of these constructs are to allow a single decision statement to include both conditional and unconditional rules.

```
if (condition)
   then apply one
   or more rules
unif
   apply this block of rules
   unconditionally
ifif
   applied only if the original
   if expression was evaulated as true
unif
   apply another block of rules
   unconditionally
else
   and this block of rules
   only if the original was false
endif
```

CAUTIONS

Conditional syntax is checked at rule load time (either server startup or reload). Basic errors such as unknown keywords and unbalanced parentheses or structure statements will be detected and reported to the corresponding Admin Menu report and to the server process log. Unless these reports are checked after modifying rule sets syntax errors may result in unexpected mappings or access.

Although the server cannot determine the correct intent of an otherwise syntactically correct conditional, if it encounters an unexpected but detectable condition during processing it aborts the request, supplying an appropriate error message.

Flow control errors (e.g. an *if* not closed by a subsequent *endif*) abort all rule processing and provide a fatal error report to the client.

7.3 Conditional Keywords

The following keywords provide a match between the corresponding request or other value and a string immediately following the delimiting colon. White space or other reserved characters may not be included unless preceded by a backslash. The actual value being used in the conditional matching may be observed using the mapping item of the WATCH facility.

Conditional Keywords

Keyword	Description
accept:	Browser-accepted content types as listed in the “Accept:” request header field. Same string as provided in CGI variable HTTP_ACCEPT.
accept-charset:	Browser-accepted character sets as listed in the “Accept-Charset:” request header field. CGI variable HTTP_ACCEPT_CHARSET.
accept-encoding:	Browser-accepted content encoding as listed in the “Accept-Encoding:” request header field. CGI variable HTTP_ACCEPT_ENCODING.
accept-language:	Browser language preferences as listed in the “Accept-Language:” request header field. CGI variable HTTP_ACCEPT_LANGUAGE.

Keyword	Description
authorization:	The raw authorization string from the request header, if any supplied. This could be simply used to test whether it has been supplied or not.
callout:	Simple boolean value. If a script callout is in progress (see "Scripting Overview, CGI Callouts".) it is true, otherwise false.
client_connect_gt:	An integer representing the current network connections (those currently being processed plus those currently being "kept alive") for the particular client represented by the current request. If greater than this value returns true, otherwise false. See Section 4.6.
cluster_member:	If the supplied node name is (perhaps currently) a member of the cluster (if any) the server may be executing on.
command_line:	The command line qualifiers and parameters used when the server image was activated.
cookie:	Raw cookie data as the text string provided in "Cookie:" request header field. CGI variable HTTP_COOKIE.
decnet:	Whether DECnet is active on the system and which version is available. This value will be 0 if not active, 4 if PhaseIV or 5 is PhaseV.
dict:	Matches the specified dictionary entry. See Section 7.5.
directory:	Tests whether the specified directory exists or not. Parameter can be a URI available for mapping by the server or a VMS file-system specification. If no parameter is supplied the request path is mapped to a file-system specification. As this conditional accesses the file-system it can be *relatively expensive in terms of server latency*.
document_root:	The DOCUMENT_ROOT CGI variable SET using the *map=root=<string>* mapping rule.
file:	Tests whether the specified file exists or not. Parameter can be a URI available for mapping by the server or a VMS file-system specification. If no parameter is supplied the request path is mapped to a file-system specification. The specification can be a directory. As this conditional accesses the file-system it can be *relatively expensive in terms of server latency*.
forwarded:	Proxy/gateway host(s) request forwarded by, as specified in request header field "Forwarded:". CGI variable HTTP_FORWARDED.
host:	The host (and optionally port) specified in request header "Host:" field. This is used by all modern browsers to provide virtual host information to the server. CGI variable HTTP_HOST.
http2:	Is true if the request is being transported using HTTP/2
instance:	Used to check whether a particular, clustered instance of WASD is available. See Section 7.3.4.
jpi_username:	The account username the server is executing as.

Keyword	Description
mapped_path:	The path resulting from mapping (phase 2 if script path involved) from which the path-translated is derived.
multihome:	Somewhat specialised conditional that becomes non-null when a client used a different IP address to connect to the service than the is bound to. Is set to the IP address the client used and may be matched using wildcard matching or as a network mask.
note:	Ad hoc information (string) provided by the server administrator using the /DO=NOTE= facility (and online equivalent) that can be used to quickly and easily modify rule processing on a per-system or per-cluster basis.
notepad:	Information (strings) stored using the SET *notepad*= mapping rule. See Section 7.3.1.
ods:	Specified as 2 or 5 (Extended File System), or as SRI file name encoding (MultiNet NFS and others) PWK encoding (PATHWORKS 4/5), ADS encoding (Advanced Server / PATHWORKS 6), SMB encoding (Samba - same as ADS).
pass:	A numeric value, 1 or 2, representing the first or second pass (if a script component was parsed) through the path mapping rules. Will be zero at other times. When the server is *reverse-mapping* a file specification will be -1.
path-info:	Path specified in the request line. CGI variable PATH_INFO.
path-translated:	VMS translation of path-info. Available after rule mapping (i.e. during authorization rule processing).
query-string:	Query string specified in request line. Same information as provided in CGI variable QUERY_STRING.
rand:	Value from a random number generator. See Section 7.3.2.
redirected:	If a request has been internally redirected (Section 12.5.2) this conditional will be non-zero. Can be used as a boolean or with a digit specified.
referer:	URL of refering page as provided in “Referer:” request header field. CGI variable HTTP_REFERER.
regex:	Simple boolean value. If configuration directive [RegEx] is enabled (and hence regular expression string matching, Chapter 6) this will be true.
remote-addr:	Client IP address. Same as provided as CGI variable REMOTE_ADDR. As with all IP addresses used for conditional testing this may be wildcard string match or network mask expressed as *address/mask-length* (see Section 7.3.7). A domain (host) name preceded by a question point may be specified (e.g. “?the.host.name”). The corresponding IP address is then looked up and compared to the client. This allows ad hoc host name based rules and is distinct from use of *remote-host*. Note that DNS lookup can introduce some latency to rule (and request) processing.
remote-host:	Client host name if name resolution enabled, otherwise the IP address (same as *remote-addr*). CGI variable REMOTE_HOST.

Keyword	Description
request:	Detect the presence of specific or unknown request fields. See Section 7.3.3.
request-method:	HTTP method (“GET”, “POST”, etc.) specified in the request line. CGI variable REQUEST_METHOD.
request-protocol:	Detect the HTTP protocol in use for the request, as “2”, “1.1”, “1.0” or “0.9”. Note that the *server-protocol* conditional will indicate 1.1 when the *request-protocol* indicates 2. The server and its applications (scripts) still treat it semantically as HTTP/1.1.
request-scheme:	Request protocol as “http:” or “https:”. CGI variable REQUEST_SCHEME.
request-uri:	The unescaped request path plus any query-string. CGI variable REQUEST_URI.
restart:	A numeric value, zero to maximum, representing the number of times path mapping has been SET *map=restart*. Can be used as a boolean or with a digit specified.
robin:	Used to check whether a particular, clustered instance of WASD is available and distribute requests to it using a round-robin algorithm. See Section 7.3.4.
script-name:	After the first pass of rule mapping (script component resolution), or during authorization processing, any script component of the request URI.
server-addr:	The service IP address. CGI variable SERVER_ADDR. This may be wildcard string match or network mask expressed as *address/mask-length*.
server_connect_gt:	An integer representing the current server network connections (those currently being processed plus those currently being “kept alive”). If greater than this value returns true, otherwise false.
server_process_gt:	An integer representing the current server requests in-progress. If greater than this value returns true, otherwise false.
server-name:	The (possibly virtual) server name. This may or may not exactly match any string provided via the *host* keyword. CGI variable SERVER_NAME.
server-port:	The (possibly virtual) server port number. CGI variable SERVER_PORT.
server-protocol:	“1.1”, “1.0”, “0.9” representing the HTTP protocol used by the request.
server-software:	The server identification string, including the version. For example “HTTPd-WASD/8.0.0 OpenVMS/AXP SSL”. CGI variable SERVER_SOFTWARE.
service:	This is the composite server name plus port as *server-name:port*. To match an unknown service use “?”.
ssl:	Simple boolean value. If request is via Secure Sockets Layer then this will be true.
syi_arch_name:	System information; CPU architecture of the server system, “Alpha”, “Itanium” or “VAX”.

Keyword	Description
syi_hw_name:	System information; hardware identification string, for example “AlphaStation 400 4/233”.
syi_nodename:	System information; the node name, for example “KLAATU”.
syi_version:	System information; VMS version string, for example “V7.3”.
tcpip:	A string derived from the UCX$IPC_SHR shareable image. It looks something like this “Compaq TCPIP$IPC_SHR V5.1-15 (11-JAN-2001 02:28:33.95)” and comprises the agent (Compaq, MultiNet, TCPware, unknown), the name of the image, the version and finally the link date.
time:	Compare to current system time. See Section 7.3.5.
trnlnm:	Translate a logical name. See Section 7.3.6.
upstream-addr:	Client proxy/accelerator IP address, when “SET CLIENT=keyword” has been applied to enable transparent up-stream proxy. Same as provided as CGI variable UPSTREAM_ADDR. As with all IP addresses used for conditional testing this may be wildcard string match or network mask expressed as *address/mask-length* (see Section 7.3.7).
user-agent:	Browser identification string as provided in “User-Agent:” request header field. CGI variable HTTP_USER_AGENT.
webdav:	Simple boolean value. If the request has been identified as WebDAV then this is true. Takes an optional parameter, “MSagent”, which is true if a Microsoft WebDAV agent has been detected.
websocket:	Simple boolean value. If a WebSocket protocol upgrade request will be true.
x-forwarded-for:	Proxied client name or address as provided in “X-Forwarded-For:” request header field. CGI variable HTTP_X_FORWARDED_FOR.

7.3.1 Notepad: Keyword

The *request notepad* is a string storage area that can be used to store and retrieve ad hoc information during path mapping and subsequent authorization processing. The notepad contents can be changed using the SET *notepad*=<*string*> or appended to using SET *notepad*=+<*string*> (Section 12.5.5). These contents then can be subsequently detected using the *notepad:* conditional keyword (or the obsolescent ’NO’ mapping conditional) and used to control subsequent mapping or authorization processing.

Notepad information persists across internal redirection processing (Section 12.5.2) and so may be used when the regenerated request is mapped and authorized. To prevent such information from unexpectedly interfering with internally redirected requests a *notepad*=“” can be used to empty the storage area.

The *dictionary* facility provides similar and arguably superior functionailtiy. See Section 7.5. In fact *notepad* is now implmented as a dictionary entry.

7.3.2 Rand: Keyword

At the commencement of each pass a new pseudo-random number is generated (and therefore remains constant during that pass). The *rand:* conditional is intended to allow some sort of distribution to be built into a set of rules, where each pass (request) generates a different one. The random conditional accepts two parameters, a *modulas* number, which is used to modulas the base number, and a *comparison* number, which is compared to the modulas result.

Hence the following conditional rules

```
if (rand:3:0)
   do this
elif (rand:3:1)
   do this
else
   do this
endif
```

would pseudo-randomly generate base numbers of 0, 1, 2 and perform the appropriate conditional block. Over a sufficient number of usages this should produce a relatively even distribution of numbers. If the modulas is specified as less than two (i.e. no distribution factor at all) it defaults to 2 (i.e. a distribution of 50%). Hence the following example should be the equivalent of a coin toss.

```
if (rand:)
   heads
else
   tails
endif
```

7.3.3 Request: Keyword

Looks through each of the lines of the request header for the specified request field and/or value. This may be used to detect the presence of specific or unknown (to the server) request fields. When detecting a specified just field the name can be provided

```
if (request:"Keep-Alive:*")
```

matching any value, or specific values can also be matched for

```
if (request:"User-Agent:*Opera*")
```

Note that all request fields known to the server have a specific associated conditional keyword (i.e. “user-agent:” for the above example). To determine whether any request fields unknown to the server have been supplied use the *request:* keyword as in the following example.

```
if (request:?)
   map * /cgi-bin/unknown_request_notify.com*
endif
```

7.3.4 Instance: and Robin: Keywords

Both of these conditionals are designed to allow the redistribution of requests between clustered WASD services. They are WASD-aware and so allow a slightly more tailored distribution than perhaps an IP package round-robin implementation might. Each tests for the current operation of WASD on a particular node (using the DLM) before allowing the selection of that node as a target. This can allow some systems to be shutting down or starting up, or have WASD shutdown for any reason, without requiring any extraordinary procedures to allow for the change in processing environment.

Instance:

The instance: directive allows testing for a particular cluster member having a WASD instance currently running. This can allow requests to be redirected or reverse-proxied to a particular system with the knowlege that it should be processed (of course there is a small window of uncertainty as events such as system shutdown and startup occur asynchronously). The behaviour of the conditional block is entirely determinate based on which node names have a WASD instance and the order of evaluation. Compare this to a similar construct using the robin: directive, as described below.

This conditional is deployed in two phases. In the first, it contains a comma-separated list of node names (that are expected to have instances of WASD instantiated). In the second, containing a single node name, allowing the selected node to be tested. For example.

```
if (instance:NODE1,NODE2,NODE3)
   if (instance:NODE1) redirect /* http://node1.domain.name/*?
   if (instance:NODE2) redirect /* http://node2.domain.name/*?
   if (instance:NODE3) redirect /* http://node3.domain.name/*?
   pass * "500 Some sort of logic error!!"
endif
pass * "503 No instance currently available!"
```

If none of the node names specified in the first phase is currently running a WASD instance the rule returns false, otherwise true. If true the above example has conditional block processed with each of the node names successively tested. If NODE1 has a WASD instance executing it returns true and the associated redirect is performed. The same for NODE2 and NODE3. At least one of these would be expected to test true otherwise the outer conditional established during phase one would have been expected to return false.

Robin:

The robin: conditional allows rules to be applied sequentially against specified members of a cluster that currently have instances of WASD running. This is obviously intended to allow a form of load sharing and/or with redundancy (not balancing, as no evaluation of the selected target's current workload is performed, see below). As with the instance: directive above, there is, of course, a small window of potential uncertainty as events such as system shutdown and startup occur asynchronously and may impact availability between the phase one test and ultimate request distribution.

This conditional is again used in two phases. The first, containing a comma-separated list of node names (that are expected to have instances of WASD instantiated). The second, containing a single node name, allowing the selected node (from phase one) to have a rule applied. For example.

```
if (robin:VAX1,ALPHA1,ALPHA2,IA64A)
   if (robin:VAX1) redirect /* http://vax1.domain.name/*?
   if (robin:ALPHA1) redirect /* http://alpha1.domain.name/*?
   if (robin:ALPHA2) redirect /* http://alpha2.domain.name/*?
   if (robin:IA64A) redirect /* http://ia64a.domain.name/*?
   pass * "500 Some sort of logic error!!"
endif
pass * "503 No round-robin node currently available!"
```

In this case round-robining will be made through four node names. Of course these do not have to represent all the systems in the cluster currently available or having WASD instantiated. The first time the 'robin:' rule containing multiple names is called VAX1 will be selected. The second time ALPHA1, the third ALPHA2, and the fourth IA64A. With the fifth call VAX1 is returned to, the sixth ALPHA1, etc. In addition, the selected nodename is verified to have a instance of WASD currently running (using the DLM and WASD's instance awareness). If it does not, round-robining is applied again until one is found (if none is available the phase one conditional returns false). This is most significant as it ensures that the selected node should be able to respond to a redirected or (reverse-)proxied requested. This is the selection set-up phase.

Then there is the selection application phase. Inside the set-up conditional other conditionals apply the selection made in the first phase (through simple nodename string comparison). The rule, in the above example a redirect, is applied if that was the node selected.

During selection set-up unequal weighting can be applied to the round-robin algorithm by including particular node names more than once.

```
if (robin:VAX1,ALPHA,VAX2,ALPHA)
```

In the above example, the node ALPHA will be selected twice as often as either of VAX1 and VAX2 (and because of the ordering interleaved with the VAX selections).

7.3.5 Time: Keyword

The *time:* conditional allows server behaviour to change according to the time of day, week, or even year. It compares the supplied parameter to the current system time in one of three ways.

1. The supplied parameter is in the form "1200-1759", which should be read as "twelve noon to five fifty-nine PM" (i.e. as a time range in minutes, generalized as *hhmm-hhmm*), where the first is the start time and the second the end time. If the current time is within that range (inclusive) the conditional returns true, otherwise false. If the range doesn't look correct false is always returned.

```
if (time:0000-0000)
   it's midnight
elif (time:0001-1159)
   it's AM
elif (time:1200-1200)
   it's noon
else
   it's PM
endif
```

2. If the supplied parameter is a single digit it is compared to the VMS day of the week (1-Monday, 2-Tuesday . . . 7-Sunday).

```
if (time:6 || time:7)
   it's the weekend
else
   it's the working week
endif
```

3. If the supplied string is not in either of the formats described above it is treated as a string match with a VMS comparision time (i.e. *yyyy-mm-dd hh-mm-ss.hh*).

```
if (time:%%%%-05-*)
   it's the month of May
endif
```

7.3.6 Trnlnm: Keyword

The *trnlnm:* conditional dynamically translates a logical name and uses the value. One mandatory and up to two optional parameters may be supplied.

```
trnlnm:logical-name[;name-table][:string-to-match]
```

The *logical-name* must be supplied; without it false is always returned. If just the *logical-name* is supplied the conditional returns true if the name exists or false if it does not. The default *name-table* is LNM$FILE_DEV. When the optional *name-table* is supplied the lookup is confined to that table. If the optional *string-to-match* is supplied it is matched against the value of the logical and the result returned.

7.3.7 Host Addresses

Host names or addresses can be an alpha-numeric string (if DNS lookup is enabled) or dotted-decimal network address, a slash, then a dotted-decimal mask. For example "131.185.250.0/255.255.255.192". This has a 6 bit subnet. It operates by bitwise-ANDing the client host address with the mask, bitwise-ANDing the network address supplied with the mask, then comparing the two results for equality. Using the above example the host 131.185.250.250 would be accepted, but 131.185.250.50 would be rejected. Equivalent notation for this rule would be "131.185.250.0/26".

7.4 Examples

The following provides a collection of examples of conditional mapping and authorization rules illustrating the use of wildcard matching, network mask matching and the various formats in which the rules may be blocked.

1. This first example shows an EXEC mapping rule being applied to a path if the request query string contains the string “example”.

```
if (query-string:*example*) exec /* /cgi-bin/example/*
```

2. In this example a block of mapping statements is processed if the virtual service of the request matches that in the conditional, otherwise the block is skipped. Note the indentation to help clarify the structure.

```
if (service:the.host.name:80)
   pass /web/* /dka0/the_host_name_web/*
   pass /graphics/* /dka100/graphics/*
   pass * "404 Resource not found."
endif
```

3. This example a series of tests allow a form of case processing where the first to match will be processed and terminate the matching process. In this case if a match does not occur rule processing continues after the *endif*.

```
if (service:the.host.name:80)
   pass /web/* /dka0/the_host_name_web/*
elif (service:next.host.name:80)
   pass /web/* /dka0/next_host_name_web/*
elif (service:another.host.name:80)
   pass /web/* /dka0/another_host_name_web/*
endif
pass /graphics/* /dka100/graphics/*
pass * "404 Resource not found."
```

4. In this (somewhat contrived) example a nested test is used to check (virtual) server name and that the request is being handled via Secure Sockets Layer (SSL) for security. If it is not an informative message is supplied. The *else* and the quotes are not really required but included here for illustration.

```
if (server-name:the.host.name)
   if (scheme:"https")
      pass /secure/* /dka0/the_host_name_web/secure/*
   else
      pass * /dka0/the_host_name_web/secure/only-via-SSL.html
   endif
endif
```

5. This would be another way to accomplish a similar objective to example 4. This uses a *negation* operator to exclude access to successive mappings if not requesting via SSL.

```
if (server-name:the.host.name)
   if (!SSL:)
      pass * /web/secure/only-via-SSL.html
   endif
   pass /secure/* /web/secure/*
   pass /other/* /web/other/*
   pass /web/* /web/web/*
   pass * "404 Resource not found."
endif
```

6. This example shows the use of a compound conditional using the AND and OR operators. It also illustrates the use of a network mask. It will exclude all access to the specified path unless the request is originating from within a specified network (perhaps an intranet) or via SSL.

```
if (path:/sensitive/* && !(remote-addr:131.185.250.0/24 || SSL:))
   pass * 404 "Access denied (SSL only)."
endif
```

7. This example illustrates restricting authentication to SSL.

```
[[*]]
["Your VMS password"=VMS]
if (!request-scheme:https)
   * r+w,#0
endif
```

8. Logical name translation may be used to dynamically alter the flow of rule interpretation.

```
if (trnlnm:HTTPD_EXAMPLE)
   pass /* /example/*
else
   pass /* /*
endif
```

9. Using a site administrator's /DO=NOTE= entry to modify rule processing. In this example the contingency of a broken back-end processor has been prepared for and a document advising clients of the temporary problem is redirected to once the administrator enters

```
$ HTTPD /DO=NOTE=PROBLEM /ALL
```

at the command-line (or via the online equivalent). Note that in this example external clients are provided with the problem advice document while internal clients may still access the back-end for troubleshooting purposes.

```
if (note:PROBLEM && !remote-addr:131.185.0.0/16)
   pass /* /problem_with_backend.html
else
   pass /* /backend/*
endif
```

Of course there are a multitude of possibilities based on this idea!

Note

The noted data persists across server startups but does not persist across system startups!

7.5 Dictionary

The per-request dictionary stores key-value string pairs related to request processing. Some entries are generated and used internally by the server and others may be inserted, value changed, removed and tested by the server admin for conditional processing purposes.

The dictionary was initially introduced as an abstraction layer between the significantly different HTTP/2 and HTTP/1.*n* header semantics and server internal processing. Its utility was then extended into configuration. It is implemented as a standard hash table with collision lists. The small cost in terms of processing is completely offset by its effectiveness.

7.5.1 Configuration Entries

Dictionary entries may be configured using the SET dict=*key*=*value* mapping rule or the DICT *key*=*value* meta keyword. These are known as *configuration entries*. Keys must begin with an alpha-numeric character but otherwise keys and values may contain any printable character, with some needing to be escaped in the text of configuration files. These are some examples of each.

```
set /example/path* dict=example_key=example\ value
set /example/path* dict=example_key="example value"
set /example/path* dict=example_key="example \"value\""

dict example_key=example\ value
dict example_key="example value"
dict example_key="example \"value\""
```

If an existing key is (re-)inserted it overwrites the old value.

An entry can have an empty value.

```
set /example/path* dict=example_key=
dict example_key=
```

An entry may be removed from the dictionary by prefixing the key name with an exclamation point.

```
set /example/path* dict=!example_key
dict !example_key
```

All configuration entries may be removed by using the exclamation point with an empty key.

```
set /example/path* dict=!
dict !
```

Note

Configuration entries persist across internal redirection processing (Section 12.5.2) and so may be used as flags or otherwise contain useful information when the regenerated request is mapped and authorized. To prevent such information from unexpectedly interfering with internally redirected requests selected or all entries can be removed in the redirected request using the above values.

7.5.2 Other Entries

As mentioned, the server generates and uses dictionary entries during request processing. There are multiple types of entry, generally insulated from each other for good reason. These entries are also available for conditional testing.

Dictionary Entries

Character	Type	Description
~	configuration	admin managed entry
$	internal	server processing
>	request	request header field
<	response	response header field

The "if (dict:*expression*)" contruct first checks for a configuration entry, then for an request header field entry, then finally for an internal entry (response entries are only available for testing after response processing begins and so not in the search list). It is also possible to test for a key of a specific type by prefixing the key name with the type character. This example shows a request header field being conditionally processed.

```
if (dict:>X-example=hello)
```

It is also possible to set an entry of a specific type by prefixing the key with the type character. For example the following will set a response header field that will be included in the header when returned to the client.

```
set /example/path* dict=<X-example="\"quoted string\""
```

Setting any non-configuration entry should only be undertaken by the literati or the brave.

7.5.3 Entry Substitution

The value of a dictionary entry can be derived in whole or part from the value of another entry or entries. This uses a somewhat familiar substitution syntax. A contrived example shows an entry being set that transfers back the request user-agent header field as a response header field.

```
set /example/path* dict=<X-user-agent=''>user-agent'
```

A similar rule can be seen applied in the WATCH report example below.

7.5.4 WATCH Dictionary

The content of a request's dictionary at significant stages of request processing can be viewed using the [x]Internal item of a WATCH report. See "WASD Web Services - Features and Facilities" .

A request dictionary WATCH point is similar to the following (end of request processing) example. Note that all of the entry types described above are present in the example, including two configured entries. Note also that two of the internal entries contain embedded line-breaks and empty lines. This is an HTTP/2 request and the expanded (HTTP/1.*n* style) *request_header* and *response_header* entries are due to WATCH items Request [x]Header and Response [x]Header also being checked. They were not required for request processing.

```
|Time_______|Module__|Line|Item|Category__|Event...|
8< snip 8<
|21:11:00.12 DICT     0836 0001 INTERNAL   DICTIONARY size:32 count:29 bytes:4193|
ENTRY 001 [005] $ {14}request_method={3}GET
ENTRY 002 [009] $ {12}request_path={15}/httpd/-/admin/
ENTRY 003 [014] > {6}accept={63}text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
ENTRY 004 [018] > {15}accept-encoding={13}gzip, deflate
ENTRY 005 [001] > {10}user-agent={116}Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWe
ENTRY 006 [007] > {15}accept-language={5}en-us
ENTRY 007 [031] > {13}authorization={30}Basic ************************
ENTRY 008 [004] > {3}dnt={1}1
ENTRY 009 [012] $ {12}request_line={28}GET /httpd/-/admin/ HTTP/1.1
ENTRY 010 [024] > {4}host={18}klaatu.private:443
ENTRY 011 [011] $ {10}http2_ping={6}44.919
ENTRY 012 [013] $ {14}request_header={372}GET /httpd/-/admin/ HTTP/1.1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-encoding: gzip, deflate
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/601.4.4 (KHTML, like
accept-language: en-us
authorization: Basic ************************
dnt: 1
host: klaatu.private:443

ENTRY 013 .012. $ {9}path_info={15}/httpd/-/admin/
ENTRY 014 [000] $ {12}query_string={0}
ENTRY 015 .004. $ {11}request_uri={15}/httpd/-/admin/
ENTRY 016 [025] ~ {7}this_is={7}a test!
ENTRY 017 [028] < {12}x-user-agent={116}Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) Apple
ENTRY 018 .018. $ {15}response_status={3}200
ENTRY 019 [026] $ {15}response_reason={2}OK
ENTRY 020 .011. < {6}server={33}HTTPd-WASD/11.0.0 OpenVMS/AXP SSL
ENTRY 021 [002] < {4}date={29}Tue, 02 Feb 2016 10:40:59 GMT
ENTRY 022 .005. < {13}accept-ranges={5}bytes
ENTRY 023 [008] < {15}accept-encoding={13}gzip, deflate
ENTRY 024 .004. < {7}expires={29}Fri, 13 Jan 1978 14:00:00 GMT
ENTRY 025 [030] < {13}cache-control={18}no-cache, no-store
ENTRY 026 .028. < {6}pragma={8}no-cache
ENTRY 027 .030. < {12}content-type={29}text/html; charset=ISO-8859-1
ENTRY 028 [006] < {14}content-length={5}15741
ENTRY 029 [019] $ {15}response_header={446}HTTP/1.1 200 OK
x-user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/601.4.4 (KHTML, lik
server: HTTPd-WASD/11.0.0 OpenVMS/AXP SSL
date: Tue, 02 Feb 2016 10:40:59 GMT
accept-ranges: bytes
accept-encoding: gzip, deflate
expires: Fri, 13 Jan 1978 14:00:00 GMT
cache-control: no-cache, no-store
pragma: no-cache
content-type: text/html; charset=ISO-8859-1
content-length: 15741

8< snip 8<
```

The first three digit number is simply the entry count in order of insertion. The second, either square bracketed or period delimited, is the hash table entry. The square brackets indicate the head of the hash table, the periods down the collision list. The single punctuation character is use to indicate and differentiate the entry type. Then are the key and equate-separated value. The brace enclosed numbers are the length of the key and value respectively.

Chapter 8
Global Configuration

The example configuration file can be used as a template.

online Web link

By default, the logical name **WASD_CONFIG_GLOBAL** locates a global configuration file. Simple editing of the configuration file changes the rules. Alternatively the Server Administration page configuration interface may be used. Changes to the global configuration file require a server restart to put them into effect.

The [IncludeFile] is a directive common to all WASD configuration, allowing a separate file to be included as a part of the current configuration. See Section 4.1.

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.

Directives requiring *periods* (timeouts, lifetimes, etc.) can be specified as a single integer (representing seconds, minutes, hours, etc., depending on the directive) or unambiguously using any one of *minutes:seconds*, *hours:minutes:seconds* or *days-hours:minutes:seconds*.

Changes to the global configuration file can be validated at the command-line before restart. This detects and reports any syntactical and fatal configuration errors but of course cannot check the *intent* of the rules.

```
$ HTTPD /DO=GLOBAL=CHECK
```

8.1 Functional Groupings

Authentication/Authorization	
[AuthBasic]	enable BASIC method
[AuthCacheEntriesMax]	maximum concurrent authentication cache entries
[AuthCacheEntrySize]	maximum authentication cache entry size in bytes
[AuthCacheMinutes]	minutes before explicitly reauthorizing user from sources
[AuthDigest]	enable DIGEST method
[AuthDigestGetLife]	DIGEST method GET lifetime
[AuthDigestPutLife]	DIGEST method PUT lifetime
[AuthFailureLimit]	retries allowed before username is marked as intruder
[AuthFailurePeriod]	period during which failure limit is applied
[AuthFailureTimeout]	period during which a recognised authentication failure is applied
[AuthRevalidateLoginCookie]	*Obsolete for WASD v10.2.1 and following.*
[AuthRevalidateUserMinutes]	minutes before use needs to reenter password
[AuthSysUafAcceptExpPwd]	accept expired SYSUAF passwords
[AuthSysUafLogonType]	LOCAL, DIALUP, NETWORK (default), REMOTE
[AuthSysUafPwdExpURL]	redirection URL is SYSUAF password if expired
[AuthSysUafUseAcme]	*Obsolete for WASD V9.3 and following.*
Buffer Sizes	
[BufferQuotaDclOutput]	allows sizing of script process SYS$OUTPUT mailbox quota
[BufferSizeDclCgiHeader]	number of bytes allocated to when processing a CGI response header
[BufferSizeDclCgiPlusIn]	number of bytes allocated to scripting process CGI-PLUSIN mailbox
[BufferSizeDclCommand]	bytes allocated to scripting process SYS$COMMAND mailbox
[BufferSizeDclOutput]	bytes allocated to scripting process SYS$OUTPUT mailbox
[BufferSizeNetFile]	maximum bytes allocated to output buffer when transfering file content

[BufferSizeNetMTU]	adjust network buffer to this value of MTU (maximum transmission unit)
[BufferSizeNetRead]	bytes allocated to client request read buffer, and to the scripting process SYS$INPUT mailbox
[BufferSizeNetWrite]	bytes allocated to client output buffer
[SocketSizeRcvBuf]	bytes allocated to a network connection receive buffer
[SocketSizeSndBuf]	bytes allocated to network connection send buffer

Content-Type

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

Directory Listing

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

[DirWildcard]	allow wildcards to be specified at all
	File Cache
[CacheChunkKBytes]	memory block allocation size
[CacheEntriesMax]	maximum number of files allowed in cache
[CacheFileKBytesMax]	maximum size of a file
[CacheFrequentHits]	identify active files
[CacheFrequentPeriod]	identify active file
[CacheGuardPeriod]	prevent early reloads
[CacheTotalKBytesMax]	maximum memory to be consumed by cache
[CacheValidatePeriod]	maximum period before the cache checks for file modification
	HTTP/2
[Http2Protocol]	enables/disables HTTP/2 on a global basis
[Http2FrameSizeMax]	maximum number of bytes in an HTTP/2 frame
[Http2HeaderListMax]	maximum number of bytes in a request or response header
[Http2HeaderTableSize]	maximum number of bytes in a request lookup table
[Http2PingSeconds]	period between RTT server-client pings
[Http2StreamMax]	number of concurrent streams (requests) permitted on a connection
[Http2InitWindowSize]	initial connection flow-control window size
	Logging
[Logging]	enable logging
[LogExcludeHosts]	hosts to be excluded from log
[LogExtend]	default allocation/extend in blocks
[LogFile]	provides part or all of log file name
[LogFormat]	nature and layout of log contents
[LogNaming]	how the log name is be constructed
[LogPeriod]	period at which new logs are created
[LogPerInstance]	create a separate log for each instance process
[LogPerService]	create a separate log for each configured service

[LogPerServiceHostOnly]	suppress service port number as component of log name
[LogWriteFail503]	generate 530 responses if the access log cannot be written

Operator Console and Log

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

Miscellaneous

[Accept]	restrictive list of host from which to accept requests
[ActivityDays]	activity graph duration
[ConnectMax]	maximum number of concurrent connections
[DNSLookupClient]	enable client host name lookup
[DNSLookupLifeTime]	host name lookup cache entry lifetime
[DNSLookupRetry]	number two second attempts to resolve client host name
[EntityTag]	provide a strong validator for file-system based resources
[GzipAccept]	advertise acceptance of GZIUP (deflated) request bodies
[GzipFlush]	period between GZIP buffer flushes
[GzipResponse]	enable GZIP (deflated) response bodies
[InstanceMax]	number of per-node server processes to maintain
[InstancePassive]	start multiple instances already in *passive* mode
[Monitor]	enable HTTPDMON data exchange
[PipelineRequests]	check for and process pipelined requests
[Port]	default port
[ProcessMax]	maximum number of concurrent requests being processed
[PutBinaryRFM]	record format of uploaded file
[PutMaxKBytes]	maximum size of a POST or PUT
[PutVersionLimit]	maximum RMS file versions retained in a POST or PUT
[RegEx]	enable regular expression matching

[Reject]	proscriptive list of hosts from which request will be rejected
[RequestHistory]	number of requests kept for request report
[SearchScript]	path to default search script
[SearchScriptExclude]	list of file extensions excluded from implied keyword search
[Service]	list of host names and/or port to create services for ***(deprecated)***
[ServiceNotFoundURL]	redirection URL when a request service is not configured
[Welcome]	list of file names that are checked for as home pages
[WWWimplied]	virtual services *host.name* and *www.host.name* are treated as synonyms

Proxy Serving

[ProxyCache]	enable proxy caching
[ProxyCacheFileKBytesMax]	maximum size of response for caching
[ProxyCacheDeviceCheckMinutes]	minutes between check of cache device usage
[ProxyCacheDeviceDirOrg]	flat 256 or 64x64 directory organization
[ProxyCacheDeviceMaxPercent]	maximum percentage of cache device used before purge
[ProxyCacheDevicePurgePercent]	during purge reduce by this many percent
[ProxyConnectPersistMax]	connection persistence for this number of connections
[ProxyConnectPersistSeconds]	connections persist for this number of seconds
[ProxyConnectTimeoutSeconds]	the proxy to origin server connect times-out after this number of seconds
[ProxyNegativeSeconds]	cache negative (failure) responses for this period
[ProxyCacheNoReloadSeconds]	prevent pragma reloads for this period
[ProxyCachePurgeList]	list of file ages used during purge
[ProxyCacheReloadList]	list of file ages before realod from source
[ProxyCacheRoutineHourOfDay]	hour of day routine cache purge occurs
[ProxyForwarded]	add “Forwarded:” to requests
[ProxyHostLookupRetryCount]	DNS resolution retry count
[ProxyReportLog]	report failures to process log
[ProxyReportCacheLog]	report cache failures to process log
[ProxyServing]	enable proxy server

[ProxyVerifyRecordMax]	enable proxy verification
[ProxyXForwardedFor]	add “X-Forwarded-For:” to requests

Reports

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

Timeout

[TimeoutHttp2Idle]	period an HTTP/2 connection remains without processing a request
[TimeoutInput]	period a connection can wait before sending request
[TimeoutNoProgress]	period a response can continue without data transfer progress
[TimeoutOutput]	period a response can continue to output
[TimeoutPersistent]	period a connection is kept active after request conclusion

Scripting

[CgiStrictOutput]	script output must be CGI compliant
[DclBitBucketTimeout]	period a script continues after a client prematurely disconnects
[DclCgiPlusLifeTime]	period of non-use before CGIplus process is deleted
[DclCleanupScratchMinutesMax]	maximum minutes between WASD_SCRATCH cleanups
[DclCleanupScratchMinutesOld]	cleanup files older than this
[DclDetachProcess]	use detached scripting processes rather than subprocesses
[DclGatewayBG]	enable raw TCP/IP socket for scripts
[DclHardLimit]	maximum number of concurrent processes
[DclScriptProctor]	proactive script and scripting environment startup

[DclScriptRunTime]	script execution environment
[DclSoftLimit]	maximum number of processes before proactive deletion begins
[DclSpawnAuthPriv]	spawn subprocesses with account's authorized privileges
[DclZombieLifeTime]	period of non-use before a CGI/CLI process is deleted
[DECnetReuseLifeTime]	period of non-use before a DECnet process is released
[DECnetConnectListMax]	maximum number of DECnet processes
[Scripting]	enables and disables all scripting

Secure Socket

[SecureSocket]	enable Secure Socket (TLS/SSL) (if built with SSL)
[SSLcert]	server certificate file
[SSLcipherList]	list of enabled/disable ciphers
[SSLinstanceCacheMax]	multiple instance shared session cache maximum number of records
[SSLinstanceCacheSize]	multiple instance shared session cache size of record
[SSLkey]	server certificate private key
[SSLoptions]	options flags
[SSLsessionCacheMax]	session cache maximum records
[SSLsessionLifetime]	session lifetime
[SSLstrictTransSec]	HSTS maxiumum age in seconds
[SSLverifyPeer]	verify client certificate
[SSLverifyPeerDataMax]	maximum kBytes of request data buffered during renegotiation
[SSLverifyPeerCAFile]	file of accepted CAs
[SSLverifyPeerDepth]	depth of certificate chain
[SSLversion]	TLS/SSL protocol versions supported

Server Side Includes

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

WebDAV	
[WebDAV]	enable WebDAV support
[WebDAVCollectionDepth]	test locking to this depth
[WebDAVlocking]	enable WebDAV locking
[WebDAVlockingTimeoutDefault]	set default lock timeout
[WebDAVlockingTimeoutMax]	set maximumg lock timeout
[WebDAVmetaDir]	location of metadata
[WebDAVquota]	enable disk quota reporting

8.2 Alphabetic Listing

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

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

   Examples:

   ```
   [Accept]
   *.www.example.com
   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 facility. Zero disables this data collection. The maximum is 28 days. 11520 bytes per day, and 80640 per week, is required to store the per-minute data.

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

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

   Examples:

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

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

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

   Examples:

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

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

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

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

   The WASD server uses a number of extensions to provide additional information. See Section 4.7.

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

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

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

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

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

   To provide additional information for correct handling of FTP transfers the transfer mode can be indicated after the content type using the FTP: keyword. One of three characters is used. An “A” indicates that this file type should be FTP transfered in ASCII mode. An “I” or a “B” indicates that this file type should be FTP transfered in Image (binary) mode.

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

To specify a VMS record format for POST or PUT files use the RFM: keyword following the content-type. This record format will always be used when creating the file. The precedence for determining the created file record format is [AddType] RFM:, then any per-path PUT=RFM= mapping rule, then [PutBinaryRFM], then a default of UDF.

```
[AddType]
.doc    application/msword  rfm:STMCR   MS Word document
```

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

   Enables or disables BASIC username authentication.

8. **[AuthCacheEntriesMax]** ***integer*** *(default: 32)*

   Maximum concurrent authentication cache entries. This needs to be sized adequately to prevent the cache from thrashing (too many attempted entries causing each to spend very little time in the cache before being replaced, only to need to be inserted again with the next attempted access).

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

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

10. **[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).

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

    Enables or disables Digest username authentication.

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

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

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

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

14. **[AuthFailureLimit]** ***integer*** *(default: 0)*

    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. Parallels the purpose of SYSGEN parameter LGI_BRK_LIM.

15. **[AuthFailurePeriod]** ***hh:mm:ss*** *(default: 00:00:00)*

    The period during which [AuthFailureLimit] is applied. Parallels the purpose of SYSGEN parameter LGI_BRK_TMO.

16. **[AuthFailureTimeout]** ***hh:mm:ss*** *(default: 00:00:00)*

The period during which which any intrusion aversion is applied. Parallels the purpose of SYSGEN parameter LGI_HID_TIM.

17. **[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.

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

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

19. **[AuthSysUafLogonType]** LOCAL | DIALUP | NETWORK | REMOTE *(default: NETWORK)*

    When SYSUAF authentication is performed *account access restrictions* are checked. By default NETWORK restrictions are used but this global configuration parameter allows another to be specified.

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

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

21. **[AuthSysUafUseAcme]**

    Obsolete for WASD V9.3 and following.

22. **[BufferQuotaDclOutput]** ***integer*** *(default: [BufferSizeDclOutput] + 256)*

    The number of bytes allocated to script SYS$OUTPUT mailbox capacity. The [BufferSizeDclOutput] sets the maximum record size and [BufferQuotaDclOutput] the total number of bytes that can be outstanding at any given time.

23. **[BufferSizeDclCgiHeader]** ***integer*** *(default: 2048)*

    The number of bytes allocated to store and process a script CGI response header.

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

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

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

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

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

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

27. **[BufferSizeNetFile]** ***integer*** *(default: none)*

The maximum bytes to be allocated to a buffer when transfering file content. For larger files this can improve both the reading of the file content from disk and when appropriately *tuned* to the local system the transmission of that content to the client, significantly increasing data rates. Limited to the $QIO maximum I/O unit of 65,535 bytes. Bigger is not always necessarily better (in the sense it always improves data rates).

28. **[BufferSizeNetMTU]** ***integer*** *(default: none)*

    This more esoteric directive attempts to minimise network buffer transmission wastage by rounding the output buffer size up to the network interface MTU (maximum transmission unit). This can provide small improvements to transmission efficiency. For example a filled buffer of 4096 with an MTU of 1500 sends two 1500 byte packets and then one of 1096 bytes, theoretically wasting some 404 bytes. A potentially better choice of buffer size would be 4500. Setting this directive to 1500 would result in the server automatically rounding a [BufferSizeNetWrite] value (for example) from 4096 up to 4500.

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

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

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

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

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

    File cache control.

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

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

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

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

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

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

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

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

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

    If a file has been hit at least [CacheFrequentHits] times in total and the last was within the period here specified it will not be a candidate for reuse. See Chapter 11.

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

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

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

    Obsolete for WASD V8.0 and following.

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

    Maximum memory allocated to the cache, in kilobytes.

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

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

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

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

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

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

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

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

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

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

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

44. **[ConnectMax]** ***integer*** *(default: 200)*

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

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

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

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

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

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

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

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

    If non-zero the CGIplus process is terminated the specified period after it last processed a request (idle for that period). Adjusting the period to suit the site allows frequently used persistent scripts and scripting engines to remain resident while more sporadically accessed ones do not remain unecessarily. If this value is zero (or unconfigured) the idle timeout is one hour.

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

    By default scripts are executed within server processes. When enabled this instructs the server to create detached processes. This side-steps the issues of having pooled process quotas and also allows non-server-account scripting and in particular “Scripting Overview, Introduction”.

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

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

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

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

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

    When enabled, non-SSL, process script CGI environments have a CGI variable WWW_GATEWAY_BG created containing the device name (BG*nnnn*:) of the TCP/IP socket connected to the client. This socket may be accessed by the script for transmission of data directly to the script bypassing the server entirely. This is obviously much more

efficient for certain classes of script. For purposes of accurate logging the server does need to be informed of the quantity of data transfered using a CGI callout. See “Scripting Environment” document.

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

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

53. **[DclScriptProctor]** ***string*** *(default: none)*

    Script proctoring proactively creates and maintains specific persistent scripts and scripting environments (RTEs). It is intended for those environments that have some significant startup latency.

    See “WASD Web Services - Scripting” for further information.

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

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

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

    The number of DCL/CGI script processing processes after which idle processes 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 processes. If these limits are not set (i.e. zero) they are calculated with [ProcessMax] using “[DclSoftLimit] = [ProcessMax]” and “[DclHardLimit] = [DclSoftLimit] + [DclSoftLimit] / 4”.

56. **[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.

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

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

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

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

59. **[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.

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

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

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

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

62. **[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).

63. **[DirLayout]** ***string*** *(default: I__L__R__S__D)*

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

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

The following shows some examples:

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

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

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

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

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

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

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

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

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

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

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

    Allows specification and display of the RMS file owner information.

68. **[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
    ```

69. **[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.

70. **[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
```

71. **[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.

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

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

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

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

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

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

75. **[EntityTag]** ENABLED | DISABLED *(default: ENABLED)*

    An entity tag is a client-opaque string used in strong cache validation. WASD generates this using the on-disk file identification (FID) and binary last-modified date-time (RDT). This is then used as a definitive identifier for a specified on-disk resource fixed in file-system space-time (hmmm, sounds like an episode of Star Trek).

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

    Specifies the **URL-format path** to an optional, error reporting SSI document or script. See Section 4.10. This path can subsequently be remapped during request processing. Optional, space-separated HTTP status codes restrict the path to those codes, with the remainder handled by server-internal reporting.

77. **[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)
    ```

78. **[GzipAccept]** ***integer*** *(default: 0)*

    Enables GZIP encoding of request bodies. See Section 4.4.

79. **[GzipFlushSeconds]** ***integer*** *(default: 0)*

    Adjusts the maxiumum period period between GZIP buffer flushes. See Section 4.4.

80. **[GzipResponse]** ***integer*****[*****integer,integer*****]** *(default: 0)*

    Enables GZIP encoding (deflation) for suitable requests and responses. Valid values are 1 for minimum compression (and minimum resource usage) through to 9 for maxiumum compression (and maximum resource usage). The value 9 is recommended. See Section 4.4.

81. **[Http2Protocol] enable | disable** *(default: disable)*)

    Enable or disable (default) HTTP/2 for all services. The default for a service follows the global setting. A service must explicitly disable HTTP/2 if that is required.

82. **[Http2FrameSizeMax]** ***integer*** *(default: 65535)*

    The maximum permitted size (in octets) of an HTTP/2 frame sent from the client.

83. **[Http2HeaderListMax]** ***integer*** *(default: 65535)*

    The maximum permitted size (in bytes) of a request header sent from the client.

84. **[Http2HeaderTableMax]** ***integer*** *(default: 4096)*

    The maximum permitted size (in bytes) of a request header compression table.

85. **[Http2PingSeconds]** ***hh:mm:ss*** *(default: 00:05:00)*

    The period at which HTTP/2 pings are sent from the server to the client to calculate the (then) Round Trip Time (RTT) of the connection.

86. **[Http2StreamMax]** ***integer*** *(default: 32)*

    Maximum number of concurrent streams (requests) supported by the connection.

87. **[Http2InitWindowSize]** ***integer*** *(default: 65535)*

    Initial flow-control window size (in bytes).

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

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

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

    Start a multiple instance server already in *passive* mode.

90. **[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 WASD_CONFIG_LOG logical name (Section 3.5).

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

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

    Example:

    ```
    [LogExcludeHosts]
    *.www.example.com
    131.185.250.*
    ```

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

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

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

    Provides some or all of the access log file name. See Section 4.12.2.

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

    Specifies one of three pre-defined formats, or a user-definable format. See Section 4.12.1.

95. **[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 4.12.5.

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

    Specifies a period at which the log file is changed. See Section 4.12.2.

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

    When multiple instances are configured (see "WASD Web Services - Features and Facilities"). create a separate log for each. This has significant performance advantages. See Section 4.12.4.

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

    When multiple services are specified (Section 4.3) a separate log file will be created for each if this is enabled. See Section 4.12.3.

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

    When generating a log name do not make the port number part of it. This effectively provides a single log file for all ports provided against a host name (e.g. a standard HTTP service on port 80 and an SSL service on port 443 would have entries in the one file). See Section 4.12.3.

100. **[LogWriteFail503]** ENABLED | DISABLED *(default: DISABLED)*

After an access log record fails to write all subsequent requests return a 503 service unavailable response until records can be successfully written again. This can be used to prevent access to server resources unless an access audit log is available.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

108. **[PipelineRequests]** ENABLED | DISABLED *(default: ENABLED)*

Pipelining refers to multiple requests being sent over an assumed persistent connection without waiting for the response from previous requests. Such behaviour with capable clients and servers can significantly reduce response latency.

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

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

110. **[ProcessMax]** ***integer*** *(default: 100)*

The maximum number of concurrent client request being processed before a "*server too busy right now ... try again shortly*" error is returned to the client. If not explicitly set this defaults to the same value as [ConnectMax]. This directive allows a larger number of persistent connections to be maintained than are concurrently being processed at any given moment.

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

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

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

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

113. **[ProxyCacheNegativeSeconds]** ***hh:mm:ss*** *(default: 00:05:00)*

Negative (unsuccessful) responses are cached for this period.

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

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

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

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

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

Organization of directories on the proxy cache device. The first provides a single level structure with a possible 256 directories at the top level and files organized immediately below these. For versions of VMS prior to V7.2 exceeding 256 files per directory, or a total of approximately 65,000 files, incurs a significant performance penalty for some directory operations. The second organization involves two levels of directory, each with a maximum of 64 directories. This allows for approximately 1,000,000 files before encountering the 256 files per directory issue.

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

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

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

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

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

Prevents pragma reloads actually retrieving the file from the source host again until the period expires. This is designed to limit concurrent or repeated reloads of files into the cache unecessarily. Thirty seconds is probably an adequate period balancing effect against a user legitimately needing to recache the document.

120. **[ProxyCachePurgeList]** ***string*** *(default: 168,48,24,8,0)*

A list of comma-separated integers representing the sequence of last accessed period in hours used during a progressive *reactive* purge.

121. **[ProxyCacheReloadList]** ***string*** *(default: 1,2,4,8,12,24,48,96,168)*

A list of comma-separated integers representing the sequence of age in hours used when determining whether a cache file's contents should be reloaded.

122. **[ProxyConnectPersistMax]** ***integer*** *(default: 100)*

The maximum number of established connections that are maintained to remote servers.

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

Period for which the established connections persist. At expiry the connection is closed.

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

Period for which the proxy server will attempt to establish a network connection to the origin (remote) server.

125. **[ProxyForwarded]** BY | DISABLED | FOR | ADDRESS *(default: DISABLED)*

BY enables the addition of a proxy request header line providing information that the request has been forwarded by another agent. The added header line would look like "Forwarded: by http://server.name.domain (HTTPd-WASD/n.n.n OpenVMS/AXP Digital-TCPIP SSL)". If the FOR variant is used the field included the host name (or ADDRESS) the request is being forwarded on behalf of, as in "Forwarded: by http://server.name.domain (HTTPd-WASD/n.n.n OpenVMS/AXP Digital-TCPIP SSL) for host.name.domain".

126. **[ProxyHostLookupRetryCount]** ***integer*** *(default: 0)*

When the server is resolving the name of a remote host the request may timeout due to up-stream DNS server latencies. This parameter allows a number of retries, at five second intervals, to be enabled.

127. **[ProxyReportLog]** ENABLED | DISABLED *(default: DISABLED)*

Enables or disables the server process log reporting siginificant proxy processing events, such as cache maintenance activity.

128. **[ProxyReportCacheLog]** ENABLED | DISABLED *(default: DISABLED)*

Enables or disables the server process log reporting of proxy caching activity.

129. **[ProxyServing]** ENABLED | DISABLED *(default: DISABLED)*

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

130. **[ProxyUnknonwRequestFields]** ENABLED | DISABLED *(default: DISABLED)*

When enabled propagates all request fields provided by the client through to the proxied server. When disabled only propagates fileds that WASD recognises.

131. **[ProxyVerifyRecordMax]** ***integer*** *(default: 0)*

Obscure functionality; see WASD Proxy Service feature.

132. **[ProxyXForwardedFor]** ADDRESS | DISABLED | ENABLED | UNKNOWN *(default: DISABLED)*

Enables the addition of a proxy request header line providing the host name on behalf of which the request is being proxied. The added header line would look like "X-Forwarded-For: host.name.domain". THE ADDRESS variant provides the IP address, and the UNKNOWN variant substitutes "unknown" for the host. This field is degined to be compatible with the *Squid* de facto standard field of the same name. Any request with an existing "X-Forwarded-For:" field has the local information appended to the existing as a comm-separated list. The first host in the field should be the original requesting client.

133. **[PutBinaryRFM]** ***FIX512*** **|** ***STM*** **|** ***STMCR*** **|** ***STMLF*** **|** ***UDF*** *(default: UDF)*

Record format for a non-text HTTP POST or PUT upload into the file-system. Has a per-path equivalent. The precedence for determining the created file record format is [AddType] RFM:, then any per-path PUT=RFM= mapping rule, then [PutBinaryRFM], then the default of UDF.

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

Maximum size of an HTTP POST or PUT method request in Kilobytes. Has a per-path equivalent.

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

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

136. **[RegEx]** ENABLED | DISABLED *(default: DISABLED)*

Enable regular expression matching. With the possibility of the reserved character "^" being used in existing mapping rules regular expression string matching (Chapter 6) is only available after enabling this directive.

The default syntax is POSIX EGREP but can be specified by substituting for ENABLED one of the following keywords; AWK, ED, EGREP, GREP, POSIX_AWK, POSIX_BASIC, POSIX_EGREP, POSIX_EXTENDED, POSIX_MINIMAL_BASIC, POSIX_MINIMAL_EXTENDED, SED. When changed from the default *enabled* (WASD) case-insensitivity is lost.

137. **[Reject]** ***host/domain name*** *(default: none)*

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

Example:

```
[Reject]
*.www.example.com
131.185.250.*
```

138. **[ReportBasicOnly]** ENABLED | DISABLED *(default: DISABLED)*

Only ever supply basic information in a report (Section 4.10).

139. **[ReportMetaInfo]** ENABLED | DISABLED *(default: DISABLED)*

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

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

The server can keep a list of the most recent requests accessible from the Server Administration page. This value determines the number kept. Zero disables the facility. Each retained request consumes 256 bytes and adds a small amount of extra processing overhead.

141. **[Scripting]** ENABLED | DISABLED *(default: ENABLED)*

Enables and disables **all** scripting mechanisms. This includes CGI and CGIplus, DECnet-based OSU and CGI, and SSI directives that DCL processes to provide <–#dcl –>, <–#exec –>, etc.

142. **[SearchScript]** ***path*** *(no default)*

Specifies the **URL-format path** to the default query-string keyword search script. This path can subsequently be remapped during request processing.

Example:

```
[SearchScript] /wasd_root/script/query
```

143. **[SearchScriptExclude]** ***list*** *(no default)*

Provides a list of file types that are excluded from an implied keyword search. This is useful for client-side (browser-side) active processing that may require a query string to pass information. This query string would normally be detected by the server and if not in a format to be meaningful to itself is then considered as an implied (HTML <ISINDEX>) keyword search, with the approriate script being activiated.

Example:

```
[SearchScriptExclude] .HTA,.HTL
```

144. **[SecureSocket]** ENABLED | DISABLED *(default: DISABLED)*

Enable the Secure Sockets Layer (SSL) Transport Layer Security (TLS) if the server has been built with that option. See “WASD Web Services - Features and Facilities” .

145. **[ServerAdmin]** ***string*** *(no default)*

Specifies the contact email address for server administration issues. Included as a “mailto:” link in the server signature if [ServerSignature] is set to *email*.

146. **[ServerAdminBodyTag]** ***string*** *(default: <BODY>)*

Specifies the HTML <BODY> tag for server administration and administration report pages. This allows some measure of control over the “look-and-feel” of page and link colour, etc.. for the administrator.

147. **[ServerReportBodyTag]** ***string*** *(default: <BODY>)*

Specifies the HTML <BODY> tag for server error and other report pages. This allows some measure of site “look-and-feel” in page colour, background, etc. to be maintained.

148. **[ServerSignature]** ENABLED | EMAIL | DISABLED *(default: DISABLED)*

The server signature is a short identifying string added to server generated error and other report pages. It includes the server software name and version, along with the host name and port of the service. Setting this to *email* makes the host name a *mailto:* link containing the address specified by the [ServerAdmin] directive.

149. **[Service]** ***string*** *(no default)* ***(deprecated)***

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

150. **[ServiceNotFoundURL]** ***string*** *(no default)*

Provides a default path for reporting a virtual host does not exist, see Section 4.3.2.

151. **[SocketSizeRcvBuf]** ***integer*** *(no default)*

Number of bytes allocated at the device-driver level for a network connection receive buffer. See Chapter 3.

152. **[SocketSizeSendBuf]** ***integer*** *(no default)*

Number of bytes allocated at the device-driver level for a network connection send buffer. Later versions of TCP/IP Services seem to have large default values for this. MultiNet and TCPware are reported to improve transfers of large responses by increasing low default values. See Chapter 3.

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

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

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

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

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

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

156. **[SSIsizeMax]** ***integer*** *(default: 0 (128kB))*

SSI source files a completely read into memory before processing. This allows the maximum size to be expanded beyond the default.

157. **[SSLcert]** ***string*** *(no default)*

TLS/SSL Configuration

See “WASD Web Services - Features and Facilities” .
Server command line /SSL= parameter equivalents override the [SSL..] directives.

TLS/SSL server certificate file path.

158. **[SSLcipherList]** ***string*** *(no default)*

A colon-separated list (OpenSSL syntax) of TLS/SSL ciphers allowed to be used by clients to connect to SSL services. The use of this parameter might allow the selection of stronger ciphers to be forced to be used or the connection not allowed to procede.

159. **[SSLinstanceCacheMax]** ***integer*** *(no default)*

TLS/SSL multiple WASD instance, shared session cache. Maximum number of shared records.

160. **[SSLinstanceCacheSize] *integer*** *(no default)*

TLS/SSL multiple WASD instance, shared session cache. Size in bytes of each individual record.

161. **[SSLkey] *string*** *(no default)*

TLS/SSL server certificate private key file path. The private key is commonly enbedded into the certificate file.

162. **[SSLoptions] *string*** *(no default)*

Alphanumeric flags supported by WASD or hexadecimal value applied to the SSL option of OpenSSL.

163. **[SSLsessionCacheMax] *integer*** *(no default)*

Single WASD instance, shared session cache. Maximum number of records. Records are dynamically sized.

164. **[SSLsessionLifetime] *hh:mm:ss*** *(no default)*

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

165. **[SSLstrictTransSec] *hh:mm:ss*** *(no default)*

When non-zero represents the number of seconds, or maximum age, of a HSTS "Strict-Transport-Security:" response header field. See "WASD Web Services - Features and Facilities" . There is an equivalent per-service directive.

166. **[SSLverifyPeer]** ENABLED | DISABLED *(default: DISABLED)*

To access this service a client must provide a verified CA client certificate.

167. **[SSLverifyPeerCAfile] *string*** *(default: none)*

Specifies the location of the collection of Certificate Authority (CA) certificates used to verify a peer certificate (VMS file specification).

168. **[SSLverifyPeerDataMax] *integer*** *(default: 1024)*

When a client certificate is requested for authentication via TLS/SSL renegotiation this is the maximum kilobytes POST/PROPFIND/PUT data buffered during the renegotiation. There is an equivalent per-service directive.

169. **[SSLverifyPeerDepth] *integer*** *(default: 0)*

Level through a certificate chain a client is verified to.

170. **[SSLversion] *string*** *(default: TLS family of protocols)*

The abbreviation for the TLS/SSL protocol version allowed to be used to connect to an SSL service. Using the directive a service may select prefered protocols.

171. **[StreamLF] *integer*** *(default: 0 (disabled))*

Enables or disables automatic conversion of VARIABLE record format documents (files) to STREAM-LF, which are much more efficient with this server. The integer is the maximum size of a file in kilobytes that the server will attempt to convert. Zero disables any conversions.

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

(Retired in v5.3, mapping SET rule provides this now, see Section 12.5.5).

173. **[TimeoutHttp2idle]** ***hh:mm:ss*** *(default: 01:00:00)*

The maximum period of time before an idle HTTP/2 connection is issued with a GOAWAY frame. An idle HTTP/2 connection is one where it has not processed a request.

174. **[TimeoutInput]** ***hh:mm:ss*** *(default: 00:01:00)*

Period allowing a connection request to be in progress without submitting a complete request header before terminating it.

175. **[TimeoutPersistent]** ***hh:mm:ss*** *(default: 0)*

The period a persistent connection with the client is maintained after the conclusion of a request. Connection persistence improves the overall performance of the server by reducing the number of discrete TCP/IP connections that need to be established.

176. **[TimeoutNoProgress]** ***hh:mm:ss*** *(default: 00:02:00)*

Period allowing request output to continue without any increase in the number of bytes transfered. This directive is targeted at identifying and eliminating requests that have stalled.

177. **[TimeoutOutput]** ***hh:mm:ss*** *(default: 00:10:00)*

Period allowing a request to be output before terminating it. This directive sets an absolute maximum time a request can continue to receive output.

178. **[WebDAV]** ENABLED | DISABLED *(default: DISABLED)*

Enable WEBdav on a server-wide basis (see “WASD Web Services - Features and Facilities”).

179. **[WebDAVlocking]** ENABLED | DISABLED *(default: DISABLED)*

Enable WebDAV locking.

180. **[WebDAVlockCollectionDepth]** ***integer*** *(default: 0)*

Ancestor directory locking depth.

181. **[WebDAVlockTimeoutDefault]** ***ddd-hh:mm:ss*** *(default: 01:00:00)*

Set default locking period.

182. **[WebDAVlockTimeoutMax]** ***ddd-hh:mm:ss*** *(default: 7-00:00:00)*

Maximum locking period.

183. **[WebDAVmetaDir]** ***string*** *(default: same as data file)*

Location of metadata files.

184. **[WebDAVquota]** ENABLED | DISABLED *(default: DISABLED)*

Enable disk quota reporting.

185. **[Welcome]** ***file.suffix*** *(no default)*

Specifies the names and order in which a directory is checked for home page files. If no home page is found a directory listing is generated.

```
[Welcome]
index.html
index.htm
home.html
home.htm
```

Dynamic home pages (script or interpreter engine driven, e.g. Perl, PHP) may be deployed using a combination of the [Welcome] and [DclScriptRunTime] directives.

```
[Welcome]
index.html
index.htm
index.php
index.pl

[DclScriptRunTime]
.PHP $CGI-BIN:[000000]PHPWASD.EXE
.PL $CGI-BIN:[000000]PERLRTE
```

186. **[WWWimplied]** ENABLED | DISABLED *(default: (DISABLED))*

When enabled considers *www.host.name* and *host.name* to be the same virtual service. If a request being processed has a virtual host of *www.host.name* and the service matching, rule matching or authentication matching process encounters a *host.name* virtual service it is considered match. A request with a virtual host of *host.name* does not match a service of *www.host.name*.

Chapter 9
Service Configuration

By default, the logical name **WASD_CONFIG_SERVICE** locates a common service configuration file. The service configuration file is optional. If the WASD_CONFIG_SERVICE logical is not defined or the file does not exist service configuration is made using the WASD_CONFIG_GLOBAL [Service] ***(deprecated)*** directives. For simple sites, those containing one or two services, the use of a separate service configuration file is probably not warranted. Once the number begins to grow this file offers a specific management interface for those services.

Precedence of service specifications:

1. /SERVICE= command line qualifier
2. WASD_CONFIG_SERVICE configuration file (if logical defined and file exists)
3. WASD_CONFIG_GLOBAL [Service] directive ***(deprecated)***

WASD *services* are also known as *virtual servers* or *virtual hosts* and can provide multiple, autonomous sites from the one HTTP server. Services can each have an independent IP address or multiple virtual sites share a single or set of multiple IP addresses. Whichever the case, the host name entered into the browser URL must able to be resolved to the IP address of an interface configured on the HTTP server system. There is no design limit to the number of services that WASD can support. It can listen on any number of IP ports and for any number of virtual services for any given port.

The server must be able to resolve its own host name/address. It is not unknown for completely new systems to have TCP/IP configuration overlooked. The server must also be able to resolve the IP addresses of any configured virtual services (Section 4.3). Failure to do so will result in the service not being configured. To avoid startup issues in the absence of a usable DNS it is suggested that for fundamental, business-critical or otherwise important services, static entries be provided in the system TCP/IP agent's local database.

Changes to the service configuration file can be validated at the command-line before restart. This detects and reports any syntactical and fatal configuration errors but of course cannot check the *intent* of the rules.

```
$ HTTPD /DO=SERVICE=CHECK
```

9.1 Specific Services

In common with other configuration files, directives associated with a specific virtual services are introduced using a double-bracket delimited host specification (Section 4.3). When configuring a service the following three components specify the essential characteristics.

- **scheme -** HTTP scheme (sometimes refered to as *protocol*). If *http:* (or omitted) it is a standard HTTP service. If *https:* an SSL service is configured.
- **host -** Host name or dotted-decimal address. If omitted, or specified as an asterisk ("*"), defaults to the system's IP host name.
- **port -** IP port the service is offered on. If omitted it defaults to 80 for an *http:* service, and to 443 for an *https:* (SSL) service.

These WASD_CONFIG_SERVICE examples illustrate the directive.

```
[[http://alpha.example.com:80]]
[[http://alpha.example.com:8080]]
```

9.2 Generic Services

A *generic* service is one that specifies a scheme and/or port but no specific host name. This is useful in a cluster where multiple systems all provide a basic service (e.g. a port 80 service). If the host name is omitted or specified as an asterisk the service substitutes the system's IP host name.

```
[[http://*:80]]
[[http://*:8080]]
```

9.3 SSL Services

See "WASD Web Services - Features and Facilities" .

Multiple virtual SSL services (https:) sharing the same certificate can essentially be configured against any host name (unique IP address or alias) and/or port in the same way as standard services (http:). Services requiring unique certificates can only be configured for the same port number against individual and unique IP addresses (i.e. not against aliases). This is not a WASD restriction, it applies to all servers for significant SSL technical reasons.

For example, unique certificates for https://www.company1.com:443/ and https://www.company2.com:443/ can be configured only if COMPANY1 and COMPANY2 have unique IP addresses. If COMPANY2 is an alias for COMPANY1 they must share the same certificate. During startup service configuration the server checks for such conditions and issues a warning about "sharing" the service with the first configured.

```
[[https://alpha.example.com]]
[[https://*:443]]
```

9.4 Administration Services

When multiple instances are configured Server Administration page access, in common with all request processing, is automatically shared between those instances. There are occasions when consistent access to a single instance is desirable. The [ServiceAdmin] directive indicates that the service port number should be used as a base port and all instances create their own service with unique port for access to that instance alone. The first instance to create an *administration service* uses the specified port, or the next successive if it's already in use, the next instance will use the next available port number, and so on. A high port number should be specified. The Server Administration page lists these services for all server instances in the cluster. This port configuration is not intended for general request activity, although with appropriate mapping and other configuration there is nothing specifically precluding the use (remembering that the actual port in use by any particular instance may vary across restarts). In all other respects the services can (and should) be mapped, authorized and otherwise configured as any other.

```
[[https://alpha.example.com]]
[ServiceAdmin]  enabled
```

9.5 IPv4 and IPv6

Both IP version 4 and 6 are concurrently supported by WASD. All networking functionality, service creation, SSL, proxy HTTP, proxy FTP and RFC1413 authorization is IPv6 enabled. If system TCP/IP services do not support IPv6 the expected error would be

```
%SYSTEM-F-PROTOCOL, network protocol error
```

during any attempted IPv6 service creation. Of course IPv4 service creation would continue as usual.

Server configuration handles the standard dotted-decimal addresses of IPv4, as well as "normal" and "compressed" forms of standard IPv6 literal addresses, and a (somewhat) standard variation of these that substitutes hyphens for the colons in these addresses to allow the colon-delimited port component of a "URL" to be resolved. Alteratively, use the de facto standard method of enclosing the IPv6 address within square brackets, followed by any port component.

IPv6 Literal Addresses

Normal	**Compressed**
1070:0:0:0:0:800:200C:417B	1070::800:200C:417B
0:0:0:0:0:0:13.1.68.3	::13.1.68.3
0:0:0:0:0:FFFF:129.144.52.38	::FFFF:129.144.52.38
hyphen-variants	
1070-0-0-0-0-800-200C-417B	1070–800-200C-417B

hyphen-variants	
0-0-0-0-0-0-13.1.68.3	–13.1.68.3
0-0-0-0-0-FFFF-129.144.52.38	–FFFF-129.144.52.38

In common with all virtual services, if a connection can be established with the system and service port the server can respond to that request. The first example binds a service to accept IPv4 connections for any address, while the second the same for IPv6 (and for IPv4 if the interface has IPv4 configuration).

```
[[https://alpha.example.com:80]]
[ServiceBind]  0.0.0.0

[[https://alpha6.example.com:80]]
[ServiceBind]  ::0
```

If a service needs to be bound to a specific IP address then that can be specified using the [ServiceBind] directive using any of the literal address formats described above.

```
[[http://alpha.example.com:80]]
[ServiceBind]  168.192.0.3

[[https://alpha6.example.com:80]]
[ServiceBind]  fe80::200:f8ff:fe24:1a22

[[https://[fe80::200:f8ff:fe24:1a22]:80]]
```

IPv6 Name Resolution

TCP/IP Services for OpenVMS *does not* provide an asynchronous name resolution ACP call for IPv6 as it does for IPv4. This means that dynamic name resolution in IPv6 environments is (currently) an issue. See the server code module [SRC.HTTPD]TCPIP6.C for further detail and workarounds. Let's hope this significant deficiency in VMS' IPv6 support is addressed sooner than later!

9.6 To www. Or Not To www.

In the twenty-first century the *www.* prefix to Web services is largely redundant. Generally *www.host.name* and *host.name* are treated as synonymous. WASD conditionals often need to distinguish precisely on the service name and in some cases this can mean a service for the *www.host.name* and the *host.name*.

The WASD global configuration directive

```
# WASD_CONFIG_GLOBAL
[WWWimplied]  enabled
```

(by default, and for backward-compatibility reasons, disabled) results in the server matching a request specifying a leading *www.* matching a virtual service identical except for the *www.*. So for the configured service.

```
[[http://the.host.name]]
```

a request to http://the.host.name/ (request header “Host: the.host.name”) or to http://www.the.host.name/ (request header “Host: www.the.host.name”) will be matched to it and allow conditionals, etc., to match to the one “the.host.name”.

9.7 Service Directives

Where a service directive has an equivalent configuration directive (e.g. error report path) the service directive takes precedence. This allows specific virtual services to selectively override the generic configuration.

Service Directives

[[virtual-service]]	scheme://host:port
[ServiceAdmin]	an *instance* Server Administration page service
[ServiceBind]	if different to host’s
[ServiceBodyTag]	<BODY> tag for server reports., etc
[ServiceClientSSLcert]	proxy SSL connect client certificate file
[ServiceClientSSLkey]	proxy SSL connect client private key file
[ServiceClientSSLcipherList]	proxy SSL connect ciphers
[ServiceClientSSLverifyCA]	verify CA of proxied requests
[ServiceClientSSLverifyCAfile]	location of proxy CA file
[ServiceClientSSLversion]	proxy SSL version to use
[ServiceErrorReportPath]	path to script, SSI or “flat” error document
[ServiceHttp2Protocol]	per-service HTTP/2 disabled
[ServiceLogFormat]	per-service access log format
[ServiceNoLog]	suppress logging
[ServiceNonSSLRedirect]	redirect non-SSL on SSL service
[ServiceProxy]	proxy service
[ServiceProxyAffinity]	make origin server “sticky”
[ServiceProxyAuth]	require proxy authorization
[ServiceProxyCache]	proxy caching
[ServiceProxyChain]	chained proxy service host
[ServiceProxyChainCred]	up-stream proxy service access credentials
[ServiceProxySSL]	provide proxy of SSL (connect:)
[ServiceProxyTunnel]	enable tunneling of octets

[ServiceRawSocket]	enable “RawSocket” scripting
[ServiceShareSSH]	share service with SSH
[ServiceSSLcert]	SSL service certificate
[ServiceSSLcipherList]	list of accepted SSL ciphers
[ServiceSSLkey]	SSL service private key
[ServiceSSLoptions]	SSL options
[ServiceSSLsessionLifetime]	SSL session lifetime
[ServiceSSLstrictTransSec]	HSTS maxiumum age in seconds
[ServiceSSLverifyPeer]	access only using verified peer certificate
[ServiceSSLverifyPeerCAfile]	location of CA file
[SSLverifyPeerDataMax]	maximum kBytes of request data buffered during renegotiation
[ServiceSSLverifyPeerDepth]	depth of certificate chain
[ServiceSSLversion]	SSL version to use

Configuration keywords equivalent to many of these WASD_CONFIG_SERVICE directives but usable against the deprecated WASD_CONFIG_GLOBAL [Service] directive and the /SERVICE qualifier are available for backward compatibility. See section *Command Line Parameters* in source file [SRC.HTTPD]SERVICE.C for a list of these keywords.

9.8 Directive Detail

Some of these directives control the behaviour of proxy services. Other directive are Secure Sockets Layer (SSL) specific.

1. **[[virtual-service]]** *(default: none)*

   Specifies the scheme, host name (or asterisk) and port of a service.

2. **[ServiceAdmin]** ENABLED | DISABLED *(default: DISABLED)*

   Marks the port as *administration* service (Section 9.4).

3. **[ServiceBind]** ***literal address*** *(default: none)*

   If the system has a multi-homed network interface this binds the service to the specific IP address and not to INADDR_ANY. Generally this will not be necessary. The literal address may be in IPv4 dotted-decimal or IPv6 normal or compressed hexdecimal.

4. **[ServiceBodyTag]** ***string*** *(default: <BODY>)*

   Specifies the HTML <BODY> tag for server error and other report pages. This allows some measure of site “look-and-feel” in page colour, background, etc. to be maintained.

5. **[ServiceClientSSL]** ENABLED | DISABLED *(default: DISABLED)*

Enables a proxy service to *originate* HTTP-over-SSL requests. This is different to the CONNECT service enabled using [ServiceProxySSL]. It allows requests to be gatewayed between standard HTTP and Secure Sockets Layer.

TLS/SSL Configuration

See “WASD Web Services - Features and Facilities” .

6. **[ServiceClientSSLcert]** ***string*** *(default: none)*

7. **[ServiceClientSSLcipherList]** ***string*** *(default: none)*

   Location of client certificate file if required to authenticate client connection.

8. **[ServiceClientSSLkey]** ***string*** *(default: none)*

   Location of client private key file if required to authenticate client connection.

   A comma-separated list of SSL ciphers to be used by the gateway to connect to SSL services. The use of this parameter might allow the selection of stronger ciphers to be forced to be used or the connection not allowed to procede.

   Note

   These *ServiceClientSSL..* directives are used to control behaviour when outgoing SSL connections are established (as with HTTP-to-SSL gatewaying). This should not be confused with verification of client certificates, which is better refered to as peer verification. See [ServiceSSLverifyPeer] and [ServiceSSLverifyPeerCAfile] directives.

9. **[ServiceClientSSLverifyCA]** ENABLED | DISABLED *(default: DISABLED)*

   Unless this directive is enabled the Certificate Authority (CA) used to issue the service’s certificate is not verified. Requires that a CA file be provided. See note in [Service-ClientSSLcipherList] above.

10. **[ServiceClientSSLCaFile]** ***string*** *(default: none)*

    Specifies the location of the collection of Certificate Authority (CA) certificates used to verify the connected-to server’s certificate (VMS file specification). See note in [ServiceClientSSLcipherList] above.

11. **[ServiceClientSSLversion]** ***string*** *(default: SSLV2|V3)*

    The abbreviation for the SSL protocol version to be used to connect to the SSL service. See note in [ServiceClientSSLcipherList] above.

12. **[ServiceErrorReportPath]** ***string*** *(default: none)*

    Specifies the **URL-format path** to an optional, error reporting SSI document or script (Section 4.10). This path can subsequently be remapped during request processing.

13. **[ServiceHttp2Protocol]** ENABLED | DISABLED *(default: ENABLED)*

    When HTTP/2 is enabled globally this allows an HTTP/1.*n*-only service to be defined.

    See “WASD Web Services - Features and Facilities” .

14. **[ServiceLogFormat]** ***string*** *(default: none)*

    Per-service access log format. See Section 4.12.1.

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

    When request logging is enabled then by default all services are logged. This directive allows logging to be suppressed for this service.

16. **[ServiceNonSSLRedirect]** [CODE][HOST-NAME | IP-ADDRESS][:PORT] *(default: none)*

    The default behaviour when a non-SSL HTTP request is begun on an SSL service is to return a 400 error and short message. This directive instead redirects the client to the specified non-SSL service. The parameter can be an optional scheme (i.e. http:// or https://), optional full host name or IP address with optional port, or only a colon-delimited port number which will redirect using the current service name. A single colon is the minimum parameter and redirects to port 80 on the current service name. The default redirect code is 307 but this can be changed by providing a leading 301 or 302.

17. **[ServiceProxy]** ENABLED | DISABLED *(default: DISABLED)*

    Enables and disables proxy request processing for this service.

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

    Uses cookies to allow the proxy server to make every effort to relay successive requests from a given client to the same origin host. This is also known as client to origin affinity or proxy affinity capability.

19. **[ServiceProxyAuth]** ***NONE*** CHAIN | LOCAL | NONE | PROXY *(default: none)*

    Makes a proxy service require authorization before a client is allowed access via it. CHAIN allows an up-stream proxy server to request authorization. LOCAL enables standard server authorization. NONE disables authorization (default). PROXY enables HTTP proxy authorization. authentication.

20. **[ServiceProxyCache]** ENABLED | DISABLED *(default: DISABLED)*

    Enables and disables proxy caching for a proxy service.

21. **[ServiceProxyChain]** ***string*** *(default: none)*

    Specifies the next proxy host if chained.

22. **[ServiceProxyChainCred]** ***string*** *(default: none)*

    Credentials for the up-stream proxy server (BASIC authentication only); in the format *username:password*.

23. **[ServiceProxyTunnel]** CONNECT | FIREWALL | RAW *(default: none)*

    Transfers octets through the proxy server. FIREWALL accepts a host and port specification before connecting. CONNECT is the traditional CONNECT protocol. RAW connects to a configured host an port.

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

    Specifies the service as providing proxying of SSL requests. This is sometimes refered as a “CONNECT” service. This proxies “https:” requests directly and is different to the HTTP-to-SSL proxying enabled using [ServiceProxyHttpSSL].

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

Enable “RawSocket” processing on the service. See the chapter on WebSocket scripting in “WASD Web Services - Scripting”

26. **[ServiceShareSSH]** ***integer*** *(default: 0 (disabled))*

    Non-zero enables service sharing with an SSH server and sets the number of seconds for input timeout.

    See “WASD Web Services - Features and Facilities” .

27. **[ServiceSSLcert]** ***string*** *(default: none)*

    Specifies the location of the SSL certificates (VMS file specification).

TLS/SSL Configuration

See “WASD Web Services - Features and Facilities” .
The [ServiceSSL..] directives override the server command line /SSL= parameter equivalents.

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

    A colon-separated list (OpenSSL syntax) of TLS/SSL ciphers allowed to be used by clients to connect to SSL services. The use of this parameter might allow the selection of stronger ciphers to be forced to be used or the connection not allowed to procede.

29. **[ServiceSSLkey]** ***string*** *(default: none)*

    Specifies the location of the SSL private key (VMS file specification).

30. **[ServiceSSLsessionLifetime]** ***hh:mm:ss*** *(no default)*

    The default maximum period for session reuse is five minutes. This is the per-service equivalent of the global directive [SSLsessionLifetime].

31. **[ServiceSSLstrictTransSec]** ***hh:mm:ss*** *(no default)*

    When non-zero represents the number of seconds, or maximum age, of a HSTS “Strict-Transport-Security:” response header field. See “WASD Web Services - Features and Facilities” . There is an equivalent global directive.

32. **[ServiceSSLverifyPeer]** ENABLED | DISABLED *(default: DISABLED)*

    To access this service a client must provide a verified CA client certificate.

33. **[ServiceSSLverifyPeerCAfile]** ***string*** *(default: none)*

    Specifies the location of the collection of Certificate Authority (CA) certificates used to verify a peer certificate (VMS file specification).

34. **[ServiceSSLverifyPeerDataMax]** ***integer*** *(default: 1024)*

    When a client certificate is requested for authentication via TLS/SSL renegotiation this is the maximum kilobytes POST/PROPFIND/PUT data buffered during the renegotiation. There is an equivalent global directive.

35. **[SSLverifyPeerDepth]** ***integer*** *(default: 0)*

    Level through a certificate chain a client is verified to.

36. **[ServiceSSLversion]** ***string*** *(default: TLS family of protocols)*

The abbreviation for the TLS/SSL protocol version allowed to be used to connect to an SSL service. Using the directive a service may select prefered protocols.

9.9 Administration

A service configuration file can be maintained using a simple text editor and WASD_CONFIG_SERVICE.

Alternatively the Server Administration facility may be used When using this interface for the first time ensure the WASD_CONFIG_SERVICE logical is correctly defined. If the file did not exist at server startup any services will have been created from the WASD_CONFIG_GLOBAL [Service] directive. These will be displayed as the existing services and will be saved to the configuration file the first time it is saved. Changes to the service configuration file require a server restart to put them into effect.

The [IncludeFile] is a directive common to all WASD configuration, allowing a separate file to be included as a part of the current configuration (Section 4.1).

Not all configuration directives may be shown depending on the type of service. For instance, unless a service is configured to provide proxy, only the [ServiceProxy] directive is displayed. To fully configure such a service enable it as proxy, save the file, then reload it. The additional directives will now be available.

There is always one empty service displayed each time the configuration menu is generated. This information may be changed appropriately and then saved to add new services to the configuration (of course, these will not be available until the server is restarted). To configure multiple new services add one at a time, saving each and reloading the file to provide a new blank service.

9.10 Examples

1. The following example shows three services being configured. The first is standard HTTP on the default (and well-known) port 80. The second is a proxy service on port 8080. This service provides both standard HTTP (with response caching enabled), SSL (connect:) access and proxy authorization required. The third service is SSL, with a host-specific certificate and key.

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

[[http://alpha.example.com:8080]]
[ServiceProxy]  enabled
[ServiceProxyAuth]  PROXY
[ServiceProxyCache]  enabled
[ServiceProxySSL]  enabled

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

2. This example shows a generic service service being configured on the well-known port 80.

```
[[http://*:80]]
```

   If a cluster of four systems, ALPHA, BETA, GAMMA and DELTA all use this configuration each will have a service accessible via the following four URLs.

```
http://alpha.example.com/
http://beta.example.com/
http://gamma.example.com/
http://delta.example.com/
```

3. The following example show two services configured against specific IP addresses. The first is an IPv4 and the second a compressed IPv6.

```
[[http://alpha.example.com:80]]
[ServiceBind]  168.192.0.3

[[https://alpha6.example.com:80]]
[ServiceBind]  fe80::200:f8ff:fe24:1a22
```

4. An *administration port* is a special configuration used to support the Server Administration facility when multiple per-node instances are configured See description above.

```
[[https://alpha.example.com:44443]]
[ServiceAdmin]  enabled
[ServiceSSLcert] WASD_ROOT:[local]alpha.pem
[ServiceSSLkey] WASD_ROOT:[local]alpha.pem
```

Chapter 10
Message Configuration

By default, the logical name **WASD_CONFIG_MSG** locates the global message configuration file. A text editor may be used to modify this configuration file. Changes require a server restart to put them into effect.

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.

Changes to the message configuration file can be validated at the command-line before restart. This detects and reports any syntactical and fatal configuration errors but of course cannot check the *intent* of the rules.

```
$ HTTPD /DO=MSG=CHECK
```

10.1 Behaviour

When an error, or other message or string, needs to be provided for the client the message database is accesssed using the following algorithm.

1. If the client request has specified a list of prefered languages using the “Accept-Language:” HTTP header field the message database is checked for support of that/those languages. If one is found then that language is used to access the message.
2. If none is found, or the client has not specified a prefered language, the client host address is checked against any list of hosts/domains provided against the language (see below). If a match occurs the specified language is used.
3. If neither of the above results in a message language the base language is used (the highest numbered language). This **must** have a complete set of messages or the server will not start!

10.2 Message File Format

By default, the system-table logical name WASD_CONFIG_MSG locates a common message file, unless an individual message file is specified using a job-table logical name. Simple editing of the message file changes the messages (after a server restart, of course). Comment lines may be included by prefixing them with the hash character (“#”), and lines continued by ensuring the last character is a backslash (“\”). The server will concurrently support an additional 3 languages to the base English (although this can be increased by recompilation :-)

Note

Care must be taken with the message file or the server may refuse to start!
Worst-case; the WASD_CONFIG_MSG.CONF message file may be copied from [EXAMPLE].

As illustrated below the message file comprises a series of sections. Directives enclosed by square-brackets provide information to the message loader.

```
# this is a comment

[version]   9.0
[language]  1  en

[general]

en 01 Sanity check failure.
en 02 String overflow.
en 03 Heap allocation failed.
en 04 calloc() failed
en 05 Request calloc() failed.
en 06 Server too busy.
en 07 Server access denied.
en 08 Facility is disabled.
en 09 Wildcard not permitted.
en 10 Directory layout problem.

[next-section, etc.]
```

The square-bracketed section headings have the following functions.

- **[version] -** Ensures the correct database version is available for the server version attempting to use it. The message file always needs checking for this version number being changed at server updates, although the version may remain fixed at a previous server version number if there have been no changes to the message database during subsequent server versions. This must be the first directive in the file.
- **[language] -** Creates space for assigning the new language's messages. The number specifies an order within the languages, each must be different, but only the lowest and highest (prefered and base respectively) have operational significance. The highest number should always be English to provide a fall-back message. A short string provides an identifier for the language. This identifier should be the same as the identifying string in the browser request "Accept-Language:" header field (e.g. "en", "se", "de", "fr", etc.) Multiple, comma-separated languages may be specified. The first is the primary language of that list and messages must be specified using that. The subsequent languages are equivalents that might be specified by the client. A wildcard may be used to match all possibilities (e.g. "de,de-*", "es,es-*"). Following the language identifier is an optional host/domain list. Multiple hosts/domains may be specified by separating each with a comma. The specifications may contain wildcards. All the [language] directives should be grouped at the start of the file immediately following the [version] directive. A character set may be associated with a particular language by specifying a *charset*= following the language string (e.g. "ru charset=koi8-r"). Setting the language's ordering number to zero disables the language completely. All messages associated with it will then be ignored.
- **[group-name] -** The messages are divided into groupings to make them easier to manage. Each group begins with the group name directive.
- **en 01 message -** Each message in a group is assigned using using this format. The string identifying the language, then the message number (the leading zero just improves the format, strictly it is not required), then the actual message itself. The message can be of arbitrary length. Long messages may be continued on following lines using the "\" continuation character.

The base language (the highest numbered, which should always be English) must have precisely the right number of messages required by the server, too few or too many and the server will not start! **Additional languages do not have to reassign every message!** The base language will supply any not assigned. A message number of zero is disabled and completely ignored.

If messages contain HTML tags that markup must not interfere with the general HTML page it is used within.

Some messages are a composite of multiple strings each of which is used on a different part of the one page (e.g. for the [upd] edit-page). Each of the strings is delimited by the vertical bar " | ". Care must be taken when customizing these strings that the overall number stays the same and that the length of each does not become excessive. Although it will not disrupt the server it may significantly disrupt the page layout.

All message numbers must be included. To provide an empty string for any one message (not recommended) provide the line with nothing following the message number.

10.3 Multiple Language Specifications

Multiple language messages can be specified in two ways:

- within the one file
- in multiple files specified by a multivalued logical name

Within The One File

Language availability is specified through the use of [Language] directives. These must be numbered from 1 to the count of those supplied. The highest numbered language must have the complete set of messages for this is the fallback when obtaining any message (this would normally be “en”). The [Language] may be specified as a comma-separated list of equivalent or similar specifications, which during request processing will be matched against a client specified list of accepted-languages one at a time in specified order. A wildcard may be specified which matches all fitting the template. In this manner a single language can be used also to match minor variants or language specification synonyms.

```
[Version]  9.0
[Language]  1  es,es-ES
[Language]  2  de,de-*
[Language]  3  en

[auth]
es 01  Habla Espanol
de 01  Sprechen Sie Deutsches
en 01  Do you speak English
.
.
.(full set of messages)
```

In the above (rather contrived) example a client request with

```
Accept-Language: es-ES,de;q=0.6,en;q=0.3
```

would have language 1 selected, a client with

```
Accept-Language: de-ch,es;q=0.6,en;q=0.3
```

language 2 selected, with

```
Accept-Language: pt-br,de;q=0.6,en;q=0.3
```

also language 2 selected, with

```
Accept-Language: pt
```

language 3 (the default) selected, etc.

Note that the messages for each language must use the *first* language specification provided in the [Language] list. In the example above all messages for language 1 would be introduced using ’es’, for language 2 with ’de’ and for language 3 with ’en’.

Multiple Files - Multivalued Logical Name

With this approach a logical name containing multiple file names is defined (more commonly described as a logical search list). The final file specified must contain the full message set. Files specified prior to this, can contain as many or as few of the full set as is desired. A [Language] number does not need to be specified as they are processed in the order the logical name specifies them in. Other language file directives are required.

The following is an example of a logical name providing the same three languages in the examples above.

```
$ DEFINE /SYSTEM WASD_CONFIG_MSG WASD_ROOT:[LOCAL]WASD_CONFIG_MSG_ES.CONF, -
                                 WASD_ROOT:[LOCAL]WASD_CONFIG_MSG_DE.CONF, -
                                 WASD_ROOT:[LOCAL]WASD_CONFIG_MSG.CONF
```

The file contents would be as follows (very contrived examples :-)

```
# WASD_CONFIG_MSG_ES.CONF
[Version]  9.0
[Language]  0  es,es-ES
[auth]
es 01  Habla Espanol
es 02  Habla Inglesi
[dir]
es 03  Habla Espanol
es 04  Habla Inglesi
```

```
# WASD_CONFIG_MSG_DE.CONF
[Version]  9.0
[Language]  0  de,de-*
[auth]
de 01  Sprechen Sie Deutsches
de 02  Sprechen Sie Englisch
[dir]
de 03  Sprechen Sie Deutsches
de 04  Sprechen Sie Englisch
```

```
# WASD_CONFIG_MSG.CONF
[Version]  9.0
[Language]  0  en
[auth]
.
.
.(full set of messages)
```

The **major advantage** of maintaining multiple files in this way is there is **no need to merge files** when a new revision is required. Just update the version number and add any new required messages to the existing secondary file.

10.4 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 WASD_ROOT:[EXAMPLE] directory.

online Web link

Note that message files can become out-of-date as server versions change, requiring modifications to the message database. Check the version information and/or comments at the top of candidate message files, however even slightly dated files may serve as a good starting point for a locale-specific message base.

Chapter 11
Cache Configuration

WASD HTTPd provides an optional, configurable, monitorable file data and revision time cache. File data, so that requests for documents can be fulfilled without reference to the underlying file system, potentially reducing request latency and more importantly improving overall server performance and system impact, and file revision time, so that requests specifying an "If-Modified-Since:" header can also benefit from the above. Files are cached using a hash derived from the VMS file-system path equivalent generated during the mapping process (i.e. represents the file name) but before any actual RMS activity. WASD can also cache the content of responses from non-file sources. This can be useful for reducing the system impact of frequently accessed, dynamically generated, but otherwise relatively static pages. These sources are cached using a hash derived from virtual service connected to and the request URI.

Why Implement Caching?

Caching, in concept, attempts to improve performance by keeping data in storage that is faster to access than the usual location. The performance improvement can be assessed in three basic ways; reduction of

- response when accessing the data (latency and transfer time)
- processing involved (CPU cycles)
- impact on the usual storage location (file system I/O)

This cache is provided to address all three. Where networks are particularly responsive a reduction in request latency can often be noticeable. It is also suggested a cache "hit" may consume less CPU cycles than the equivalent access to the (notoriously expensive) VMS file system. Where servers are particularly busy or where disk subsystems particularly loaded a reduction in the need to access the file system can significantly improve performance while simultaneously reducing the impact of the server on other system activities.

A comparison between cached and non-cached performance is provided in in the "Server Performance" section.

Terminology

Term	Description
hit	Refers to a request path being found in cache. If the data is still valid the request can be supplied from cache.
flushing	Occurs when the cache becomes full, with older, less frequently used cache entries being removed from the cache and replaced by other files.
loading	Refers to reading the contents of a file into cache memory.
permanent	These entries are loaded once and remain in the cache until it is explicitly purged by the administrator or the the server is restarted. They are not flushed or revalidated.
revalidate	Compare the cache entrys size and modification date-time to the file it represents in the file-system. Obviously a difference indicates the content has changed.
valid	The file from which the cached data was originally read has not had its revision date changed (the implication being the file contents have not changed).
volatile	Entries have the original file periodically checked for modification and are reloaded if necessary. They can also be flushed if demand for space requires it.

11.1 Non-File Content Caching

The WASD cache was originally provided to reduce file-system access (a somewhat expensive activity under VMS). With the expansion in the use of dynamically generated page content (e.g. PHP, Perl, Python) there is an obvious need to reduce the system impact of some of these activities. While many such responses have content specific to the individual request a large number are also generated as general site pages, perhaps with simple time or date components, or other periodic information. Non-file caching is intended for this type of dynamic content.

Revalidation of non-file content is fraught with a number of issues and so is not provided. Instead the cache entry is flushed on expiry of the [CacheValidateSeconds], or as otherwise specified by path mapping, and the request is serviced by the content source (script, PHP, Perl, etc.) with the generated response being freshly cached. All of the considerations described in Section 11.4 apply equally to file and non-file content.

Controlling Non-File Content Caching

Determining which non-file content is cached and which not, and how long before flushing, is done using mapping rules (Section 12.5.5). The source of non-file cache content is specified using one or a combination of the following SET rules against general or specific paths.

cache=[no]cgi from Common Gateway Interface (CGI) script response
cache=[no]file from the file system (default and pre-8.4 cache behaviour)
cache=[no]net caches the full data stream irrespective of the source
cache=[no]nph full stream from Non-Parse Header (NPH) script response
cache=[no]query cache requests with query strings (**use with care**)

cache=[no]script both CGI and NPH script responses
cache=[no]ssi from Server-Side Includes (SSI) documents

A good understanding of site requirements and dynamic content sources, along with considerable care in specifying cache path SETings, is required to cache dynamic content effectively. It is especially important to get the content revalidation period appropriate to the content of the pages. This is specified using the following path SETings.

cache=expires=0 cancels any expiry
cache=expires=DAY expires when the day changes
cache=expires=HOUR when the clock hour changes
cache=expires=MINUTE when the clock minute changes
cache=expires=<hh:mm:ss> expires after the specified period in the cache

For example. To cache the content of PHP-generated home pages that contain a time-of-day clock, resolving down to the minute, would require a mapping rule similar to the following.

```
set /**/index.php cache=cgi cache=expires=minute
```

11.2 Permanent and Volatile

The WASD file cache provides for some resources to be permanently cached while others are allowed to be moved into and out of the cache according to demand. Most sites have at least some files that are fundamental components of the site's pages, are rarely modified, commonly accessed, and therefore should be permanently available from cache. Other files are modified on a regular or ad hoc basis and may experience fluctuations in demand. These more volatile resources should be cached based on current demand.

Volatile caching is the default with the site administrator using mapping rules to indicate to the server which resources on which paths should be permanently cached (Section 11.5).

Although permanent and volatile entries share the same cache structure and are therefore subject to the configuration's maximum number of cache entries, the memory used store the cached file data is derived from separate pools. The total size of all volatile entries data is constrained by configuration. In contrast there is no configuration limit placed on the quantity of data that can be cached by permanent entries. One of the purposes of the permanent aspect of the cache is to allow the site administrator considerable discretion in the configuration of the site's low-latency resources, no matter how large or small that might be. Of course there is the ultimate constraint of server process and system virtual memory limits on this activity. It should also be kept in mind that unless sufficient physical memory is available to keep such cached content in-memory the site may only end up trading file-system I/O for page file I/O.

11.3 Cache Suitability Considerations

A cache is not always of benefit! the cost may outweigh the 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 accessible data to the overhead of cache management, due to the constant and continuous flushing and reloading of cache data. This server's cache is no different, it will only improve performance if the site experiences some consistency in the files requested. For sites that have only a small percentage of files

being repeatedly requested it is probably better that the cache be disabled. The other major consideration is available system memory. On a system where memory demand is high there is little value in having cache memory sitting in page space, trading disk I/O and latency for paging I/O and latency. On memory-challenged systems cache is probably best disabled.

To help assessment of the cache's efficiency for any given site monitor the Server Administration facility's cache report.

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

- **Activity Data**

  This summarizes the cache search behaviour, in particular that of the hash table.

  The "searched" item, indicates the number of times the cache has been searched. Most importantly, this may include paths that can never be cached because they represent non-file requests (e.g. directory listings). Requests involving scripts, and some others, never attempt a cache search.

  The "hit" item, indicates the number of times the hash table directly provided a cached path. This is very efficient.

  The "miss" item, indicates the number of times the hash table directly indicated a path was not cached. This is decisive and is also very efficient.

  The "collision" item, indicates the number of times multiple paths resolved to the same hash table entry. Collisions require further processing and are far less efficient. The sub-items, "collision hits" and "collision misses" indicate the number of times that further processing resulted in a found or not-found cache item.

  A large number of cache misses compared to searches may only indicate a large number of non-cacheable requests and so depending on that further datum is not of great concern. A large proportion of collisions (say greater than 12.5%) is however, indicating either the hash table size needs increasing (1024 should be considered a minimum) or the hashing algorithm in the software need reviewing :-)

- **Files Data**

  This summarizes the site's file request profile.

  With the "loads not hit" item, the count represents the cumulative number of files loaded but never subsequently hit. If this percentage is high it means most files loaded are never hit, indicating the site's request profile is possibly unsuitable for caching.

  The item "hits" respresents the cumulative, total number of hits against the cumulative, total number of loads. The percentage here can range from zero to many thousands of percent :-) with less than 100% indicating poor cache performance and from 200% upwards better and good performance. The items "1-9", "10-99" and "100+" show the count and percentage of total hits that occured when a given entry had experienced hits within that range (e.g. if an entry has had 8 previous hits, the ninth increments the "1-9" item whereas the tenth and eleventh increments the "10-99" item, etc.)

Other considerations also apply when assessing the benefit of having a cache. For example, a high number and percentage of hits can be generated while the percentage of "loads not hit" could be in the also be very high. The explanation for this would be one or two frequently requested files being hit while most others are loaded, never hit, and flushed as other files request cache space. In situations such as this it is difficult to judge whether cache processing is improving performance or just adding overhead.

11.4 Cache Content Validation

The cache will automatically revalidate the volatile entry file data after a specified number of seconds ([CacheValidateSeconds] configuration parameter), by comparing the original file revision time to the current revision time. If different the file contents have changed and the cache contents declared invalid. If found invalid the file transfer then continues outside of the cache with the new contents being concurrently reloaded into the cache. Permanent entries are not subject to revalidation and the associated reloading.

Cache validation is also always performed if the request uses "Cache-Control:" with *no-cache*, *no-store* or *max-age=0* attributes (HTTP/1.1 directive), or if a "Pragma: no-cache" field (HTTP/1.0 directive). These request directives are often associated with a browser agent *reload page* function. Hence there is no need for any explicit flushing of the cache under normal operation. If a document does not immediately reflect any changes made to it (i.e. validation time has not been reached) validation (and consequent reload) can be "forced" with a browser reload. Permanent entries are also not subject to this source of revalidation. The configuration directive [CacheGuardPeriod] limits this form of revalidation when used within the specified period since last revalidated. It has a default value of fifteen seconds.

If a site's contents are relatively static the validation seconds could be set to an extended period (say 3600 seconds, one hour) and then rely on an explicit "reload" to force validation of a changed file.

The entire cache may be purged of cached data, both volatile and permanent entries, either from the Server Administration facility or using command line server control.

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

11.5 Cache Configuration

The cache is controlled using WASD_CONFIG_GLOBAL configuration file and WASD_CONFIG_MAP mapping file directives. A number of parameters control the basics of cache behaviour.

- **[Cache]** enables and disables caching.
- **[CacheEntriesMax]** and **[CacheTotalKBytesMax]** provide growth limits to cache expansion. Maximum entries limits the number of files loaded into the cache before entries begin to be reused (flushing the original contents). Maximum total kilobytes allocated to the cache provides a ceiling on the memory consumed. These parameters operate to limit each other (i.e. if one reaches its limit before the other, the other will not grow further either).

- **[CacheFileKBytesMax]** provides a limit on file size (in kilobytes). Files larger than the specified limit will not be cached. This may be overridden on a per-path basis using the *set cache=max=<integer>* mapping rule (see below).
- **[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.
- **[CacheGuardPeriod]** prevents browser initiated content revalidation described above (Section 11.4). It is provided to help limit unnecessary file-system activity. The default is fifteen seconds.
- **[CacheEntriesMax]** *(obsolete)*
- **[CacheValidateSeconds]** The interval after which a cache entry's original, content revision time is revalidated against the file's current revision time. If not the same the contents are declared invalid and reloaded. Setting this to a greater period reduces disk I/O but revised files may not be obvious within an acceptable timer unless a revalidation is forced with a *reload*. Permanent entries are not subject to validation.

Mapping Rules

Mapping rules (Section 12.5.5) allow further tailoring of cache behaviour based on request (file) path. Those files that should be made permanent entries are indicated using the *cache=perm* directive. In the following example all files in the WASD runtime directories (directory icons, help files, etc.) are made permanent cache entries at the same time the path is mapped.

```
pass /*/-/* /wasd_root/runtime/*/* cache=perm
```

Of course, specified file types as well as specific paths can be mapped in this way. Here all files in the site's /help/ path are made permanent entries except those having a .PS type (PostScript documents).

```
set /help/* cache=perm
set /help/*.ps cache=noperm
```

The configuration directive [CacheFileKBytesMax] puts a limit on individual file size. Those exceeding that limit are considered too large and not cached. It is possible to override this general constraint by specifying a maximum size (in kilobytes) on a per-path basis.

```
set /help/examples*.jpg cache=max=128
set /cai/*.mpg cache=max=2048 cache=perm
```

Caching may be disabled and/or enabled for specified paths and subpaths.

```
set /web/* cache=none
set /web/icons/* cache
```

11.6 Cache Control

The cache may be enabled, disabled and purged from the Server Administration facility. In addition the same control may be exercised from the command-line 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.

11.7 Circumventing The Cache

There are often good reasons for bypassing or avoiding the cache. For instance, where a document is being refreshed within the cache revalidation period specified by [CacheValidateSeconds] (Section 11.4). There are two mechanisms available for bypassing or invalidating the file cache.

1. This directs the server to always get the file from the file-system.

   ```
   SET /path/not/to/cache/* cache=none
   ```

2. Specify a version component when requesting the file. WASD never caches a file if the request contains a version component. It does not need to be a full version number, a semi-colon is sufficient. For example:

   ```
   /wasd_root/robots.txt;
   ```

Chapter 12

Request Processing Configuration

By default, the logical name **WASD_CONFIG_MAP** locates a common mapping rule file. Simple editing of the mapping file and reloading into the running server changes the processing rules. The [IncludeFile] is a directive common to all WASD configuration, allowing a separate file to be included as a part of the current configuration (Section 4.1).

Mapping rules are used for a number of different request processing purposes.

1. To map a request *path* onto the VMS file system. That is, to map from web-space into file-space.
2. To map from file-space back into web-space. There is often not a one-to-one correspondance between file specifcations and web paths.
3. To process a request path according to specified criteria resulting in an effective path that is different to that supplied with the request.
4. 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.
5. To conditionally map to different end-results based on one or more criteria of the request or environment.
6. To provide differing virtual sites depending on the actual service accessed by the client.

Mapping is basically for server-internal purposes only. The only time the path information of the request itself is modified is when a script component is removed. At all other times the path information remains unchanged. Path authorization is always applied to the path supplied with the request.

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.

Changes to the mapping configuration file can be validated at the command-line before reload or restart. This detects and reports any syntactical and fatal configuration errors but of course cannot check the *intent* of the rules.

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

A server's currently loaded mapping rules may also be interrogated from the Server Administration menu (see "WASD Web Services - Features and Facilities").

12.1 Rule Interpretation

The rules are scanned from first towards last, until a matching final rule is encountered (PASS, EXEC, SCRIPT, FAIL, REDIRECT, UXEC and USER) when the mapping pass concludes. Non-final rules (MAP and SET) perform the appropriate action and continue to the next rule. One, two or more passes through the rules may occur due to implicit processing (if the path contains a script component) or by explicit restart (SET *map=restart*).

String Matching

The basis of path mapping is string pattern matching, comparing the request specified path, and optionally other components of the request when using configuration conditionals (Chapter 7), to a series of patterns, usually until one of the patterns matches, at which stage some processing is performed. Both wildcard and regular expression based pattern matching is available. All rules have a *template* (string pattern to match against the path). Some rules have a *result* (how to restructure the components matching from the template).

- The **template** may contain one or more asterisk ("*") wildcard symbols, or a regular expression with optional grouping operators. This is pattern matched against the request path (Chapter 6). If neither is present then the path must match the *template* exactly.
- The **result** may contain one or more asterisk ("*") substitution symbols. The *result* wildcards are expanded to replace the matching strings of the respective *template* wildcards or pattern groups. Specified wildcard substitution is available (Section 6.4). Characters represented by wildcards in the *template* not represented by a corresponding wildcard in the *result* are ignored. Non-wildcard *result* characters are directly inserted in reconstructed path. Non-wildcard characters in the *template* are ignored. If the *result* contains no wildcards it completely replaces the URL path.

Virtual Servers

As described in Section 4.3 virtual service syntax may be used with mapping rules to selectively apply rules to one specific service. If virtual services are configured rule interpretation sees only rules common to all services and those specific to its own service (host address and port). In all other aspects rule interpretation applies as described above.

Processing Overhead

Naturally, each rule that needs to be processed adds a little to consumed CPU, introduces some latency, and ultimately reduces throughput. The test-bench has shown this to be acceptably small compared to the overall costs of responding to a request. Using the ApacheBench tool on a COMPAQ Professional Workstation XP1000 with 2048MB, VMS V8.3, TCP/IP Service 5.7 and WASD v10.1, with a simple access to /wasd_root/exercise/0k.txt showed approximately 744 requests/second throughput using the following mapping file.

```
pass /wasd_root/exercise/*
```

After adding various quantities of the same intervening rule

```
pass /wasd_root/example/*
pass /wasd_root/example/*
   .
   .
   .
pass /wasd_root/example/*
pass /wasd_root/exercise/*
```

the following results were derived.

Mapping Overhead

Intervening Rules	Requests/S	Throughput
0	744	baseline
100	701	-5.8%
200	665	-10.6%
500	571	-23.3%
1000	461	-38.4%

Although this is a fairly contrived set-up and actual real-world rule-sets are more complex than this, even one hundred rules is a very large set, and it does indicate that for all intents and purposes mapping rules may be used to achieve desired objectives without undue concern about impact on server throughput.

12.2 VMS File System Specifications

The VMS file system in mapping rules is always assumed to begin with a device or concealed device logical. Specifying a Master File Directory (MFD) component, the [000000] is completely optional, although always implied. The mapping functions will always insert one if required for correct file system syntax. That is, if the VMS file system mapping of a path results in a file in a top-level directory an MFD is inserted if not explicitly present in the mapping. For example, both of the following paths

```
/dka100/example.txt
/dka100/000000/example.txt
```

would result in a mapping to

```
DKA100:[000000]EXAMPLE.TXT
```

The MFD is completely optional when both specifying paths in mapping rules and when supplying paths in a request. Similarly, when supplying a path that includes directory components, as in

```
/dka100/dir1/dir2/example.txt
/dka100/000000/dir1/dir2/example.txt
```

both mapping to

```
DKA100:[DIR1.DIR2]EXAMPLE.TXT
```

LOGICAL NAMES

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. You must be able to perform a

```
$ DIRECTORY logical-name:[000000]
```

to be able to use the specification as a WASD mapping rule.

Concealed device logicals are created using the following syntax:

```
$ DEFINE LOGICAL_NAME device:[dir1.dir2.]
$ DEFINE LOGICAL_NAME /TRANSLATION=CONCEALED physical_device:[dir1.dir2.]
$ DEFINE LOGICAL_NAME /TRANSLATION=CONCEALED -
physical_device1:,physical_device2:
$ DEFINE LOGICAL_NAME /TRANSLATION=CONCEALED -
physical_device3:[dir1.dir2.],physical_device4:[dir1.dir3.]
```

The logical name may be multi-valued and provided the DIRECTORY command can be used successfully with them (as described above) should be amenable to WASD directory listing producing equivalent results.

12.3 Traditional File Specifications (ODS-2)

For ODS-2 volumes, when during rule mapping of a path to a VMS file specification an RMS-invalid character (e.g. “+”) or syntax (e.g. multiple periods) is encountered a dollar symbol is substituted in an attempt to make it acceptable. This functionality is often useful for document collections imported to the local web originating from, for instance, a Unix site that utilizes non-VMS file system syntax. The default substitution character may be changed on a per-path basis using the SET rule (Section 12.5.5).

12.4 Extended File Specifications (ODS-5)

OpenVMS Alpha V7.2 introduced a new on-disk file system structure, ODS-5. This brings to VMS in general, and WASD and other Web servers in particular, a number of issues regarding the handling of characters previously not encountered during (ODS-2) file system activities. ODS-2 and ODS-5 volumes should be automatically distinguished by the server however it is possible to *force* interpretation using a path mapping rule (Section 12.5.5).

12.4.1 Characters In Request Paths

There is a standard for characters used in HTTP requests paths and query strings (URLs). This includes conventions for the handling of reserved characters, for example “?”, “+”, “&”, “=” that have specific meanings in a request, characters that are completely forbidden, for example white-space, control characters (0x00 to 0x1f), and others that have usages by convention, for example the “~”, commonly used to indicate a username mapping. The request can otherwise contain these characters provided they are URL-encoded (i.e. a percentage symbol followed by two hexadecimal digits representing the hexadecimal-encoded character value).

There is also an RMS standard for handling characters in extended file specifications, some of which are forbidden in the ODS-2 file naming conventions, and others which have a reserved meaning to either the command-line interpreter (e.g. the space) or the file system structure (e.g. the “:”, “[”, “]” and “.”). Generally the allowed but reserved characters can be used in ODS-5 file names if escaped using the “^” character. For example, the ODS-2 file name “THIS_AND_THAT.TXT” could be named “This^_^&^_That.txt” on an ODS-5 volume. More complex rules control the use of character combinations with significance to RMS, for instance multiple periods. The following file name is allowed on an ODS-5 volume, “A-GNU-zipped-TAR-archive^.tar.gz”, where the non-significant period has been escaped making it acceptable to RMS.

Of course characters absolutely forbidden in request paths must still be URL-encoded, the most obvious example is the space. RMS will accept the file name “This^ and^ that.txt” (i.e. containing escaped spaces) but the request path would need to be specified as “This%20and%20that.txt”.

Unlike for ODS-2 volumes, ODS-5 volumes do not have “invalid” characters, so no processing is performed to ensure RMS compliance.

12.4.2 File Name Ambiguity

ODS-5 allows for some file name ambiguity in web-space.

For example the file name

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

would be presented to the client as

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

which when provided in a URL as

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

and translated from that URL into the file specification

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

of course will not be able to be accessed.

In addition, the two files

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

are distinct in the file-system, independently parsed from the directory structure, presented by a web directory listing (and WebDAV resource property list) as consecutive entries having the same name, with only the accessible file name actually available.

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

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

While these sorts of situations are corner-cases it is best to try and avoid *interesting* file names that can challenge the rather convoluted VMS file-system environment.

12.4.3 Characters In Server-Generated Paths

When the server generates a path to be returned to the browser, either in a viewable page such as a directory listing or error message, or as a part of the HTTP transaction such as a redirection, the path will contain the URL-encoded equivalent of the *canonical form* of an extended file specification escaped character. For example, the file name "This^_and^_ that.txt" will be represented by "This%20and%20that.txt".

When presenting a file name in a viewable page the general rule is to also provide this URL-equivalent of the unescaped file name, with a small number of exceptions. The first is a directory listing where VMS format has been requested by including a version component in the request file specification. The second is in similar fashion, but with the *tree* facility, displaying a directory tree. The third is in the navigation page of the *UPDate* menu. In all of the instances the canonical form of the extended file specification is presented (although any actual reference to the file is URL-encoded as described above).

12.5 Rules

These are the categories of mapping rules.

- Map paths to the file system, and to other paths:
  MAP
  PASS
  FAIL
  REDIRECT
  USER
- Provide access to scripting:
  EXEC
  SCRIPT
  UXEC
- Sets characteristics against particular paths:
  SET

12.5.1 MAP, PASS, FAIL Rules

1. **map *template result***

   If the URL path matches the template, substitute the *result* string for the path and use that for further rule processing. Both template and result paths must be absolute (i.e. begin with "/").

2. **pass *template***
   pass *template result*
   pass *template "999 message text"*

   If the URL path matches the template, substitute the result if present (if not just use the original URL path), processing no further rules.

   The *result* should be a either a physical VMS file system specification in URL format or an *HTTP status-code message* (see below). If there is a direct correspondance between the *template* and *result* the result may be omitted.

   Note

   The PASS directive is also used to *reverse-map* VMS file specifications to the URL path format equivalent. See Section 12.6.

   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 "/").

12.5.2 REDIRECT Rule

1. **redirect *template result***

   If the URL path matches the template, substitute the *result* string for the path. Process no further rules. Redirection rules can provide result URLs in one of a number of formats, each with a slightly different behaviour.

   1. The *result* can be a full URL ("http://host.domain/path/to/whatever"). This is used to redirect requests to a specific service, usually on a another host. A *result* may or may not contain a fixed query string ("/path/to/whatever?one=two").
   2. If the scheme (e.g. "http:") is omitted the scheme of the current request is substituted. This allows HTTP requests to be transparently redirected via HTTP and HTTPS (SSL) requests via HTTPS (e.g. "//host.domain/path/to/whatever", note the leading double-slash).

3. In a similar fashion both the scheme and the host name may be omitted (e.g. "///path/to/whatever", note the leading triple-slash). The server then substitutes the appropriate request scheme and host name before returning the redirection to the client.

4. If the scheme is provided but no host component the current request's host information is substituted and the redirection made using that (e.g. "https:///secure/path/to/whatever". This effectively allows a request to be redirected from standard to SSL, or from SSL to standard HTTP on the same server.

5. Alternatively, it may be just a path ("/path/to/whatever", a single leading slash), which will cause the server to internally generate an entire new request structure to process the new path (i.e. request redirection is not returned to the client).

Note

Internal redirection (as this is termed) is a fundamental mechanism available with WASD to completely change the request path and/or query string components for the request - transparently to the client. It is essentially a complete rewrite of the request.

6. Full request URI rewriting (path and any query string) is available using the *map=uri* path SETing (Section 12.5.5).

7. Only if the last character in the *result* is a question mark ("?") will any query string in the original be propagated into the redirection URL (that is the original request "/original/test.txt?plus=query" is mapped using "redirect /original/* /path/to/*?" does the resulting URL become "/path/to/test.txt?plus=query").

12.5.3 USER Rule

The USER rule maps a VMS user account default device and directory (i.e. *home* directory) into a request path. That is, the base location for the request is obtained from the VMS systems SYSUAF file. This is usually invoked by a request path in the form "/~username/", see Section 12.10 for more detailed information.

1. **user *template result***

   If the path matches the template then the result is substituted, with the following conditions. At least one wildcard must be present. The first wildcard in the result substitutes the username's home directory into the path (in place of the "~username"). Any subsequent wildcard(s) substitute corresponding part(s) of the original path.

   If the user DANIEL's default device and directory were

   ```
   USER$DISK:[DANIEL]
   ```

   the following rule

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

   would result in the following path being mapped and used

```
/user$disk/daniel/www/
```

Note

Accounts that possess SYSPRV, are CAPTIVE, have been DISUSERED or that have expired passwords will not be mapped. A "directory not found" error report is returned.

12.5.4 EXEC/UXEC and SCRIPT, Script Mapping Rules

Also see "WASD Web Services - Scripting" for further information.

The EXEC/UXEC and SCRIPT directives have the **variants EXEC+/UXEC+ and SCRIPT+**. These behave in exactly the same fashion and simply mark the rule as representing a CGIplus script environment.

The EXEC/UXEC rules maps script **directories**.

The SCRIPT rules maps script **file names**. It behaves a little differently to the EXEC rule, essentially supplying in a single rule the effect of a MAP then an EXEC rule.

Both rules must have a *template* and *result*, and both must end in a wildcard asterisk. The placement of the wildcards and the subsequent functionality is slightly different however. Both template and result paths must be absolute (i.e. begin with "/").

1. **exec *template result***

   The EXEC rule requires the *template*'s asterisk to immediately follow the slash terminating the directory specification containing the scripts. The script name follows immediately as part of the wildcard-matched string. For example:

   ```
   exec /htbin/* /wasd_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 accessible, a possible security concern if script management is distributed.

2. **exec *template (run-time-environment)result***

   A variation on the "exec" rules allows a Run-Time Environment (RTE) to be mapped. An RTE is a persistant scripting environment not unlike CGIplus. The essential difference is an RTE provides an environment in which a variety of scripts can be run. It is often an interpreter, such as Perl, where the advantages of persistance (reduced response latency and system impact) are available. For more information on RTEs and how they operate see the "WASD Scripting Environment" document.

   The RTE executable is specified in parentheses prefixed to the mapping result, as show in this example:

   ```
   exec /pl-bin/* (cgi-bin:[0000000]perlrte.exe)/wasd_root/src/perl/*
   ```

3. **script *template result***

The SCRIPT rule requires the *template*'s asterisk to immediately follow the *unique string* identifying the script in the URL path. The wildcard-matched string is the following path, and supplied to the script. For example:

```
script /conan* /wasd_root/script/conan*
```

If the URL path matches the template, the result becomes the URL format physical VMS file specification for the DCL procedure of the script to be executed (the default file extension of ".COM" is not required). What remains of the original URL path is used to create the path information. Process no further rules.

Note

The wildcard asterisk is best located immediately after the unique script identifier. In this way there does not need to be any path supplied with the script. If even a slash follows the script identifier it may be mapped into a file specification that may or may not be meaningful to the script.

Hence, the SCRIPT rule will match only the script specified in the *result*, making for finely-granular scripting at the expense of a rule for each script thus specified. It also implies that only the script name need precede any other path information.

It may be thought of as a more efficient implementation of the equivalent functionlity using two CERN rules, as illustrated in the following example:

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

4. **uxec *template result***

   The UXEC rule is an analog to the EXEC rule, except it is used to map user scripts. It requires two mapping asterisks, the first for the username, the second for the script name. It must be used in conjunction with a SET *script=as=~* rule. For example:

```
SET   /~*/cgi-bin/*  script=as=~
UXEC  /~*/cgi-bin/*  /*/www/cgi-bin/*
```

   For further information see User Account Scripting and the "Scripting Overview, Introduction".

Script Location

It is conventional to locate script images in WASD_ROOT:[AXP-BIN] or WASD_ROOT:[VAX-BIN] (depending on the platform), and procedures, etc. in WASD_ROOT:[CGI-BIN]. These multiple directories are accessible via the single search list logical CGI-BIN.

Script files can be located in area completely outside of the WASD_ROOT tree. Two approaches are available.

1. Modify the search list CGI-BIN to include the additional directories. Only should be done with extreme care.
2. Use mapping rules to make the script accessible. This can be done by using the EXEC or SCRIPT rule to specify the directory directly as in these examples

```
exec /mycgi-bin/* /site_local_scripts/bin/*
script /myscript* /web/myscripts/bin/myscript.exe*
```

or by using the MAP rules to make a hierarchy of script locations obvious and accessible, as in this example

```
map /cgi-bin/myscripts/* /cgi-bin_myscripts/*
exec /cgi-bin_myscripts/* /web/myscripts/bin/*
```

EXEC Directories and EXEC Files

Generally directories are specified as locations for script files. This is the more common application, with the EXEC rules used as in this example

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

Mapping a file type into an EXEC behaviour is also supported. This allows all files within the specified path and with the matching file suffix (extension) to be activated as scripts. Of course a script runtime must be available for the server to be able activate it. The following example demonstrates mapping all files ending in .CGI in the /web/ tree as executable scripts.

```
exec /web/*.cgi* /web/*.cgi*
```

WARNING

Remember scripts are **executables**. Enabling scripting in a general user area allows **any** user to write and execute any script, by default under the scripting account. Deploy with discretion.

12.5.5 SET Rule

The SET rule does not change the mapping of a path, it just sets one or more characteristics against that path that affect the subsequent processing in some way. It is a general purpose rule that conveniently allows the administrator to tell the server to process requests with particular paths in some ad hoc and generally useful fashion. Most SET parameters are single keywords that act as boolean switches on the request, some require parameter strings. Multiple space-separated parameters may be set against against the one path in a single SET statement.

- **ACCEPT=LANG=<*parameter*> -** Allows a path to be marked for language-variant document processing.

Rule	Description
ACCEPT=LANG= DEFAULT=<*language*>	sets the default language
ACCEPT=LANG= CHAR=<*character*>	sets the delimiting character
ACCEPT=LANG= VARIANT=<*name*> | <*type*>	allows the alternate file-type variant to be specified

Rule	Description
ACCEPT=LANG= (DEFAULT=<*language*>, CHAR=<*character*>)	sets both (etc.)
NOACCEPT=LANG	disables language variant processing (on a subtree for example)

For detailed configuration information see Section 4.8.

- **ALERT[=<*keyword*>] -** Marks a path as being of specific interest. When a request containing this path is detected by the server it puts a message into the the server process log and perhaps of greater immediate usefulness the increase in alert hits is detected by HTTPDMON and this (optionally) provides an audible alert. The following is ordered according to how early in processing the alert is signalled.

Rule	Description
ALERT=MAP	generates this alert immediately after path mapping (i.e. before the request actually begins being processed)
ALERT=AUTH	after authorization (i.e. when any remote username has been resolved)
ALERT=<*integer*>	if the response HTTP status matches the specific integer
ALERT=END	at the conclusion of process (the default)
NOALERT	cancels alerts on this path (perhaps subpath)

- **AUTH=<*keyword*> -** Changes the specified characteristic during subsequent authorization processing.

Rule	Description
[NO]AUTH=ALL	All requests matching this path must have been subject to authorization or fail with a forbidden status. This is a per-path requivalent of implementing the per-server /AUTHORIZE=ALL policy, and is a little “belt and braces” in a certain sense, but does permit a site to further avoid unintended information leakage (in this case through the failure ensure a given path has authorization).
[NO]AUTH=ONCE	If a request path contains both a script component and a resource component by default the WASD server makes sure both parts are authorized before allowing access. This can be disabled using this path setting. When this is done only the original request path undergoes authorization.

Rule	Description
AUTH=REVALIDATE=<*hh:mm:ss*>	Authorization is cancelled and the client requested to reenter the username and password if this period expires between authorized requests. Overrides configuration directive [AuthRevalidateUserMinutes].
AUTH=SYSUAF= PWDEX-PURL=<*string*>	Parallels the [AuthSysUafPwdExpURL] configuration directive, allowing it to be set on a per-path or virtual service basis.

- **CACHE=<*keyword*> -** The default is to cache files (when caching is enabled, Chapter 11).

Rule	Description
CACHE=NONE	disables caching of files matching this rule
CACHE=EXPIRES=0	cancels previous mapped expiry
CACHE=EXPIRES=DAY	expires on change of day
CACHE=EXPIRES=HOUR	expires on change of hour
CACHE=EXPIRES=MINUTE	expires on change of minute
CACHE=EXPIRES=<*period*>	sets the expiry period for the entry
CACHE=GUARD=<*period*>	sets the guard period (no reload) for the cache entry
CACHE=MAX=<*integer*>	cache files up to this many kilobytes (overrides [CacheFileK-BytesMax])
CACHE=[NO]CGI	cache CGI-compliant (script) responses
CACHE=[NO]FILE	cache files matching this rule (the default)
CACHE=[NO]NET	cache any network output
CACHE=[NO]NPH	cache NPH (non-parse-header script) responses
CACHE=[NO]SCRIPT	cache both CGI and NPH responses
CACHE=[NO]SSI	cache SSI document responses
CACHE=[NO]QUERY	cache (script) regardless of containing a query string
CACHE=[NO]PERM	permanently cache these files

- **CGIPLUSIN=<*keyword*> -** Provides control over how CGIplus records on the CGI-PLUSIN stream are carriage controlled and how the stream is terminated. A little esoteric certainly; ask Alex Ivanov ;-)

Rule	Description
CGIPLUSIN=CC=NONE	no carriage control

Rule	Description
CGIPLUSIN=CC=LF	each record has a trailing line feed (0x0a)
CGIPLUSIN=CC=CR	a trailing carriage return (0x0d)
CGIPLUSIN=CC=CRLF	a trailing line feed then carriage return (0x0d0a)
CGIPLUSIN=[NO]EOF	the end of the record stream is indicated using an end-of-file

- **CGIPREFIX=<*string*> -** CGI environment variable names are by default prefixed with "WWW_". This may be changed on a per-path basis using this SET rule. To remove the prefix altogether for selected scripts use "CGIprefix=".
- **CHARSET=<*string*> -** This setting allows overriding of the server default ([CharsetDefault] configuration parameter) content-type character set (in the response header) for text files (plain and HTML). A string is required as in the following example, "charset=ISO-8859-5".
- **CLIENT=<*keyword*> -** Client IP address data is often used during conditional mapping and as represented by CGI variable data in scripts and interpreter environments. This setting allows an up-stream proxy/accelerator to provide the actual client IP address via request header and have that data substitute for the instrinsic IP address of the up-stream proxy. This provides a level of transparency to server processing via such a proxy.

Rule	Description
CLIENT=FORWARDED	Substitute the (first) address from the "Forwarded": request header. Return a 403 status if no "Forwarded:" header present.
CLIENT=IF=FORWARDED	As above but the absence of a "Forwarded:" request header is not fatal.
CLIENT=LITERAL=<*string*>	Substitue the following string. Intended for testing purposes.
CLIENT=RESET	Reset the substituted client data to the original (up-stream proxy).
CLIENT=XFORWARDEDFOR	Substitute the (first) address from the "X-Forwarded-For": request header. Return a 403 status if no "X-Forwarded-For:" header present.
CLIENT=IF=XFORWARDEDFOR	As above but the absence of a "X-Forwarded-For:" request header is not fatal.

- **CONTENT=<*string*> -** The content-type of a file is normally determined by the file's type (extension). This setting allows files matching the template to be returned with the specified content-type. The content-type must be specified as a parameter, e.g. "content=application/binary".

- **CSS=<*URI*> | <*URL*> -** Provides a path (URI) or full URL to a stylesheet for a WASD-generated page (e.g. a directory listing). Adds a

  ```
  <LINK REL="stylesheet" TYPE="text/css" HREF="uri">
  ```

  to the page HTML header.
- **DICT=<*key*>=<*value*> -** Set a dictionary entry. See Section 7.5.
- **DIR=<*keyword*> -** Allows directory listing to be controlled on a per path basis. These parallel the coresponding configuration [Dir..] directives.

Rule	Description
DIR=[NO]ACCESS	allows directory listing
DIR=ACCESS=SELECTIVE	allows directory listing if the directory contain the file .WWW_BROWSABLE
DIR=DELIMIT=<*keyword*>	header, footer, both, none
DIR=[NO]ILINK	icon plain-text link can be disabled
DIR=[NO]IMPLIEDWILDCARD	add wildcards if not in path
DIR=SORT=<*column*>	pre-sort a listing
DIR=STYLE=<*keyword*>	set the style of a directory listing "ANCHOR" the v8.2 thru v10.3 WASD style "DEFAULT" the current WASD style (v10.4 and later) "HTDIR" Alex Ivanov's HTdir style "ORIGINAL" WASD traditional style (before v8.2) "SORT" listing sortable on column "TABLE" using HTML table layout (v10.4 and later) "*above*2" any of the above without horizontal rules
DIR=TARGET=<*string*>	open the file in another window "_blank" opens the file in a new window or tab "_self" in the same frame "_parent" in the parent frame "_top" in the full body of the window "*framename*" in the named frame
DIR=THESE=<*filespec*>	restrict listing to specified filename(s)

Rule	Description
DIR=TITLE=<*keyword*>	format the title of the window (tab) "0" (digit zero) suppress any title "1..99" where 1 is the top-level directory (device), 2 is the second-level directory, 3 . . . 99 the current directory "DEFAULT" the default for the directory *style* "OWNER" the VMS account owning the directory "REMOTE" the remote user name (for X509 authentication the certificate common-name)
DIR=VERSIONS=<*integer*>	list the specified maximum number of file versions, or if an asterisk all versions
DIR=[NO]WILDCARD	allow a directory listing to be "forced" by including wildcards in the path

- **[NO]EXPIRED -** This setting allows files in the specified paths to be sent pre-expired. The browser should always then reload them whenever accessed.
- **HTML=<*keyword*>=<*string*> -** Allows the <BODY> tag, and header and/or footer characteristics and text to be added to selected server generated pages such as directory listings and error messages.

Rule	Description
HTML=BODYTAG=	specifies the page <BODY> tag characteristics (e.g. html=bodytag="BGCOLOR=#ffffff")
HTML=HEADER=	the page header text
HTML=HEADERTAG=	the <TD> tag characteristics of the header table (e.g. html=headertag="BGCOLOR=#cccccc")
HTML=FOOTER=	the page footer text
HTML=FOOTERTAG=	the <TD> tag characteristics of the footer table

The *headertag* and *footertag* directives also allow the full table tag to be specified, allowing greater flexibility with these parts of the page (e.g. html=footertag="<TABLE BORDER=1 CELLPADDING=10 CELLSPACING=0><TR><TD BGCOLOR=#cccccc>".

- **HTTP=<*parameter*> -** Explicitly sets an aspect of the HTTP request header.

Rule	Description
HTTP=ACCEPT-CHARSET=<*string*>	the "Accept-Charset:" field

Rule	Description
HTTP=ACCEPT-LANGUAGE=<*string*>	the “Accept-Language:” field

- **HTTP2=<*parameter*> -** Controls an aspect of an HTTP/2 connection, or initiates an action on that connection.

Rule	Description
HTTP2=PROTOCOL=1.1	send the client an HTTP_1_1_REQUIRED error whcich should cause it to re-request as HTTP/1.1
HTTP2=SEND=GOAWAY[=<*integer*>]	send the client a connection GOAWAY frame with optional error number
HTTP2=SEND=PING	send the client an HTTP/2 ping
HTTP2=SEND=RESET[=<*integer*>]	send the client a stream (request) reset (close) with optional error number
HTTP2=WRITE=*LOW* | *NORMAL* | *HIGH*	this stream (request) will write to the network at the specified priority relative to other data on the connection

- **INDEX=<*string*> -** This setting provides the “Index of” (directory listing) format string for directory paths matching the template. It uses the same formatting as can be supplied with a URL and overrides any query string passed via any URL.
- **[NO]LOG -** When server access logging is enabled the default is to log all requests. The NOLOG setting suppresses logging for requests involving the specified path template.
- **MAP=<*parameter*> -** Controls aspects of the mapping processing itself (from that point in the rules onwards of course).

Rule	Description
[NO]MAP=ELLIPSIS	By default the use of the VMS file specification ellipsis wilcard (“...”) is not allowed. This enables this for the path specified. Use with caution.
[NO]MAP=ONCE	Normally, when a script has been identified during mapping, the resultant path information is also mapped in a second pass. This can be suppressed by SETing the path as MAP=ONCE. The resultant path is then given to the script without further processing.

Rule	Description
MAP=RESTART	Causes an immediate change to the order of rule processing. Instead of the next rule, the first rule in the configuration is processed. This is intended to remove the need for copious repetition in the rule set. A common or set of common processing blocks can be established near the start of the rule set and be given requests from processing points further down in the rules. It is intended to be used only once or perhaps twice and will abort the request if it occurs too often. Can be detected using the *restart:* conditional (Section 7.3). Use with caution! Injudicious use would make unexpected mappings expected!
[NO]MAP=ROOT=<*string*>	Prefixes the results of following rules with the specified path so that they are all subordinate to it. This also populates the DOCUMENT_ROOT CGI variable. See Document Root.
[NO]MAP=SET=IGNORE	All path SETings following an IGNORE are completely ignored (not applied to the mapping or request characteristics) until a subsequent NOINGORE is encountered.
[NO]MAP=SET=REQUEST	All path SETings following a NOMAP=SET=REQUEST are only applied to the mapping and not to the request's characteristics until a subsequent MAP=SET=REQUEST is encountered. Intended for use during callouts. These can be detected using the *callout:* conditional (Section 7.3).
[NO]MAP=URI	Normally mapping is performed on the request path. This SETing replaces the path with the full, raw, request URI (undecoded path plus any query string). This allows subsequent mapping rules to be applied to the full URI and therefore path components to be remapped into query components, and query components into path components (using specified substitution, see Section 6.4).

- **NOTEPAD=[+]<*string*> -** The *request notepad* is a string storage area that can be used to store and retrieve ad hoc information during path mapping and subsequent authorization processing. Multiple *notepad=string* set against the one request override previous settings unless preceded by a leading plus symbol, when it appends. These contents then can be subsequently detected using the *notepad:* conditional keyword (Section 7.3.1) or the obsolescent 'NO' mapping conditional.
- **ODS=<*keyword*> -** Directs the server on how to process file names for naming conventions other than ODS-2 (the default). Be sure to add an asterisk at the end of the specific ODS path otherwise only the top-level will set!

Rule	Description
ODS=2	is basically redundant, because if a path is not indicated as anything else it is assumed to be ODS-2. This can be used for clarity in the mapping rules if required.
ODS=5	is used to indicate that a particular path maps to files on an ODS-5 (EFS) volume and so the names may comply to extended specifications. This changes the way file names are processed, including for example the replacement of invalid RMS characters (see below).
ODS=ADS	is used to process file names that are encoded using the Advanced Server (PATHWORKS 6) schema.
ODS=NAME=*8BIT* | *UTF8* | *DEFAULT*	When a file is PUT (created) using WebDAV or upload, for non-7bit ASCII file names use native ODS-5 8bit syntax (default) or UTF-8 encoded character sequences.
ODS=PWK	is used for processing file names encoded using the PATHWORKS 4/5 schema.
ODS=SMB	is a synonym for ODS=ADS and makes clear the path is also being served by Samba.
ODS=SRI	for file names encoded using the SRI schema (used by MultiNet and TCPware NFS, FTP and other utilities).

- **QUERY-STRING=<*string*> -** Set the request's query string to that specified in the directive. Overloads any current query string. Specify URL-encoded if the characters require it.
- **PROXY=<*parameter*> -** Sets an aspect of proxy request processing.

Rule	Description
PROXY=[NO]AFFINITY	sets client to origin server affinity.
PROXY=BIND=<*ip-address*>	makes outgoing proxy requests appear to originate from this IP address. Must be an address that the media can be bound to.
PROXY=CHAIN=<*host:port*>	makes outgoing proxy requests chain to this up-stream proxy server.
PROXY=CHAIN=CRED=<*username:password*>	provides proxy authentication credentials to an up-stream proxy server.

Rule	Description
PROXY=FORWARDED	controls generatation a proxy "Forwarded:" request field. This optional field contains information on the proxy server and as a further option the client name or IP address. "PROXY=NOFORWARDED" disables "PROXY=FORWARDED[=BY]" contains the *by* component. "PROXY=FORWARDED=FOR" contains *by* and the *for* components (client host name). Also used with WASD_TUNNEL (proxy tunneling). "PROXY=FORWARDED=ADDRESS" contains *by* and the *for* components (client host address). Also used with WASD_TUNNEL (proxy tunneling).
PROXY=HEADER=<*name*>[=<*string*>]	removes or sets the value of the specified proxied request header. Examples: "PROXY=HEADER=referer" would remove the "Referer:" header field from the proxied request "PROXY=HEADER=referer=http://whatever/" would set the "Referer:" header field to the specified URL "PROXY=HEADER=user-agent=Nosey 1.0" would set the "User-Agent:" header field to the "Nosey 1.0"
PROXY=REVERSE=[NO]AUTH	suppresses propogation of any "Authorize" header.
PROXY=REVERSE= LOCATION=<*string*>	rewrites the matching "Location:" header field URL of a 302 response from an internal, reverse-proxied server.
PROXY=REVERSE= [NO]VERIFY	sets a specialized authorization capability. See WASD_ROOT:[SRC.HTTPD]PROXYVERIFY.C for further information.
PROXY=TUNNEL=REQUEST=<*string*>	allows the originating end of a WASD tunnel to specify an HTTP request line or even request header to be provided to the tunnel target end when the connection is established.
PROXY=UNKNOWN	causes the server to propagate all request field provided by the client to the proxied server (by default WASD only propagates those it recognises).

Rule	Description
PROXY=XFORWARDEDFOR=<*keyword*>	controls generation of a proxy "X-Forwarded-For:" request field. This optional field (a defacto standard originally from the *Squid* caching package) contains the name or IP address of the proxied client. "PROXY=NOXFORWARDEDFOR" disables "PROXY=XFORWARDEDFOR[=ENABLED]" enables "PROXY=XFORWARDEDFOR=ADDRESS" field contains client host address "PROXY=XFORWARDEDFOR=UNKNOWN" field contains *unknown* for the client host name

- **PUT=<*parameter*> -** Per-path control over HTTP POST or PUT request body.

Rule	Description
PUT=MAX=<*integer*> | *	Maximum number of kilobytes allowed for a request body, if "*" then effectively unlimited (per-path equivalent of the global directive [PutMaxKbytes]).
PUT=RFM=*FIX512* | *STM* | *STMCR* | *STMLF* | *UDF*	When a request body is uploaded into the file-system and the content-type is not text this determines the file record format. The precedence for determining the created file record format is [AddType] RFM:, then any per-path PUT=RFM= mapping rule, then [PutBinaryRFM], then the default of UDF.

- **[NO]PROFILE -** When using the server /PROFILE qualifier allow or disallow the authentication profile when assessing access for a specific path. Must be used in conjunction with an equivalent authorisation rule (WASD_CONFIG_AUTH) flagging the profile use against an equivalent path (see "WASD Web Services - Features and Facilities").
- **REGEX=<*keyword*> -** The default regular expression syntax is POSIX EGREP but can be specified on a per-path basis using one of the following keywords; AWK, ED, EGREP, GREP, POSIX_AWK, POSIX_BASIC, POSIX_EGREP, POSIX_EXTENDED, POSIX_MINIMAL_BASIC, POSIX_MINIMAL_EXTENDED, SED. When changed from the default *enabled* (WASD) case-insensitivity is lost. Reset expression syntax to global default using *regex=default*. **Note** that SETing the regular expression syntax in this way adds overhead as each expression then needs to be regex-compiled with each match.
- **REPORT=<*parameter*> -** This setting allows error and other server-generated reports for any specified path to changed between *detailed* and *basic* (Section 4.10.1).

Rule	Description
REPORT=BASIC	include less detail in error message

Rule	Description
REPORT=DETAILED	includes more detail
REPORT=TUNNEL	brief, non-HTML error messages suitable for proxy tunnel
REPORT=4<*nn*>=<*nnn*>	maps one 400 class HTTP status to another (to conceal the true origins of some error messages)

- **RMSCHAR=<*character*> -** This setting applies to ODS-2 paths (the default) only. Paths SET as ODS-5 do not have this applied. During rule mapping of a path to a VMS file specification, if an RMS-invalid character (e.g. "+") or syntax (e.g. multiple periods) is encountered a dollar symbol is substituted in an attempt to make it acceptable. This setting provides an alternate substitution character. Any general RMS-valid character may be specified (e.g. alpha-numeric, '$', '-' or '_', although the latter three are probably the only REAL choices). A single character is required as in the following example, "RMSchar=_".
- **RESPONSE=HEADER=<*parameter*> -** changes the way in which a response header is generated by the server.

Rule	Description
RESPONSE=GZIP=<*keyword*>	controls generation of GZIPed response bodies (Section 4.4) "ALL" suitable responses "NONE" of the responses "*integer*" kilobytes, responses known to be this size or greater
RESPONSE=HEADER=BEGIN	suppresses the response header terminating empty line so that the file or other resource can supply additional header fields. It, of course, must supply the header-terminating empty line before beginning to supply the response body.
RESPONSE=HEADER=FULL	reverts to normal response header generation behaviour.
RESPONSE=HEADER=NONE	suppresses the normal response header generation. It is considered the file or other resource contains and will supply the full HTTP response (in a non-parse-header script fashion).
RESPONSE=HEADER=ADD=<*string*>	appends the specified string to the response header. Of course the string should be a legitimate HTTP response field and value line. This mapping can be used to add a particular response directive to matching requests.

- **RESPONSE=VAR=<*parameter*> -** Where a response is being provided from a variable-length record file each record should be terminated as follows.

Rule	Description
RESPONSE=VAR=CRLF	carriage-return+line-feed (0x0D then 0x0A)

Rule	Description
RESPONSE=VAR=LF	line-feed (0x0A) character (default)
RESPONSE=VAR=NONE	nothing should be appended to the record

- **SCRIPT=<*parameter*> -** Provides controls over various aspects of the scripting environment.

  For scripting detail see the "WASD Web Services - Scripting" document.

Rule	Description
SCRIPT=AS=<*parameter*>	for non-server account scripting this rule allows the user account to be either explicitly specified or substituted through the use of the tilde character "~" or the dollar "$".
SCRIPT=BIT-BUCKET=<*hh:mm:ss*>	specifies the period for which a script continues to execute if the client disconnects. Overrides the WASD_CONFIG_GLOBAL [DclBitBucketTimeout] configuration directive.
[NO]SCRIPT=BODY=DECODE	instructs the server to decode (un-chunk and/or un-GZIP) an encoded request body before transfering it to the script. The script must be aware of this and change its processing accordingly. See Section 4.4.
SCRIPT=CONTROL=<*string*>	Supply the specified string to the CGI processor as if the a script had provided it using a "Script-Control:" response header field.

SCRIPT=COMMAND=<*string*>	allows additional parameters and qualifiers to be passed to the script activation command line. First parameter must be an asterisk to use the server resolved script command. If the first parameter is not an asterisk it substitutes for the script activation verb. Subsequent parameters must be as they would be used on the command line. The following setting
	`set /cgi-bin/example* script=command="* \ /ONE /TWO=THREE FOUR"`
	would result in the hypothetical script being command-line activated
	`$ EXAMPLE /ONE /TWO=THREE FOUR`

SCRIPT=CPU=<*hh:mm:ss*>	specifies that the server should not allow the script to use more than the specified quantity of CPU time. This is approximate, due to the way the server administers scripting. It can serve to prevent scripts from consuming indefinite quantities of system resources.

Rule	Description
SCRIPT=DEFAULT=<*string*>	sets the default directory for the script environment (a SET DEFAULT immediately prior to script activation). This can be suppressed (for backward compatibility purposes) using a "#" as the target directory. This string is reflected in CGI variable SCRIPT_DEFAULT so that CGIplus script and RTE engines can be informed of this setting for a particular script's environment. Unix syntax paths may also be specified. If the default begins with a "/" character the SET DEFAULT is not performed but the SCRIPT_DEFAULT variable is set appropriately allowing the equivalent of a *chdir()* to be performed by the scripting environment.
[NO]SCRIPT=FIND	by default the server always confirms the existance and accessability of a script file by searching for it before attempting to activate it. If it does not exist it reports an error. It may be possible a Run-Time Environment (RTE) may require to access its own script file via a mechanism available only to itself. The server script search may be disabled by SETing the path as *nofind*, for example "script=nofind". The script path and filename is directly passed to the RTE for it to process and activate.
SCRIPT=LIFETIME=<*hh:mm:ss*>	provides a per-path (and hence per-script) value for a script process *zombie* (idle scripting process) or idle CGIplus and RTE process lifetime. This per-path SETing overrides the respective [DclZombieLifeTime] and [DclCGIplusLifeTime] global directives.

SCRIPT=PARAM=<*name=value*>	allows non-CGI environment variables to be associated with a particular script path. The name component becomes a variable containing the specified value passed to the script. Multiple, comma-separated *name=value* pairs may be specified. The value may be quoted. The following path setting
	`set /cgi-bin/example* \`
	`  script=params=(first=one,second="Two (and Three)")`
	would result in additional CGI variables available to the script
	`WWW_FIRST == "one"`
	`WWW_SECOND == "Two (and Three)"`
	Multiple *script=params* set against the one request override previous settings unless the parameters are specified with a leading plus symbol, as in
	`set /cgi-bin/example* \`
	`  script=params=+(third=three,fourth="number 4")`

Rule	Description
[NO]SCRIPT=PATH=FIND	directs the server to check for and report if the file specified in the path does not exist before activating the script process. Normally this would be left up to the script.
[NO]SCRIPT=QUERY=NONE	saves a small amount of overhead by suppressing the decomposition of any query string into key or form fields for those environments that do this for themselves.
[NO]SCRIPT=QUERY=RELAXED	normally when the CGI variables are being prepared for a script and the query string is parsed an error is reported if it uses *x-www-form-urlencoded* format and the encoding contains an error. However some scripts use non-strict encodings and this rule allows those scripts to receive the query strings without the server complaining first.
[NO]SCRIPT=SYNTAX=UNIX	provides the SCRIPT_FILENAME and PATH_TRANSLATED CGI variables in Unix file-system syntax rather than VMS file-system syntax (i.e. /DEVICE/dir1/dir2/file.type rather than DEVICE:[DIR1.DIR2]FILE.TYPE).
[NO]SCRIPT=SYMBOL= TRUNCATE	allows otherwise aborted script processing to continue. Script CGI variables are provided using DCL symbols. With VMS V7.3-2 and later symbol capacity is in excess of 8000 characters. For VMS V7.3-1 and earlier it has a limit of around 1000 characters. If a symbol is too large the server by default aborts the request generating a 500 HTTP status. If the above mapping is made (against the script path) excessive symbol values are truncated and such symbol names placed into a special CGI variable named SERVER_TRUNCATE.

- **[NO]SEARCH=NONE -** Do not activate the automatic document search script for any query strings associated with this path.
- **SERVICE=<*string*> -** When mapping is concluded move the request to this virtual service or to the first virtual service matching a wildcarded specification.
- **SSI=<*parameter*> -** Controls aspects of Server-Side Include engine behaviour.

Rule	Description
[NO]SSI=PRIV	SSI documents cannot contain privileged directives (e.g. <– #exec ... –>) unless owned by SYSTEM ([1,4]) or are in path set as allowing these directives. Use SSI=priv to enable this, NOSSI=priv to disable. **Caution:** these SSI directives are quite powerful, use great care when allowing any particular document author or authors to use them.

Rule	Description

SSI=EXEC=<*string*>	where <string> is a comma-separated list of the #dcl parameters permitted for the path allows fine-grained control of what capabilities are enabled. The parameter "#" enables SSI on a per-path basis.
	`ssi=exec=say,show`
	`ssi=exec=#`

- **SSLCGI=<*keyword*> -** Enables and sets the type of CGI variables used to represent a Secure Sockets Layer (SSL) CGI variables.

  When enabling these variables it is advised to increase the WASD_CONFIG_GLOBAL [BufferSizeDclCommand] and [BufferSizeCgiPlusIn] directives by approximately 2048.

Rule	Description
NOSSLCGI	disables the facility
SSLCGI=none	disables the facility
SSLCGI=Apache_mod_SSL	provides Apache mod_ssl style variables
SSLCGI=Apache_mod_SSL_extens	provides variables representing X509 V3 extensions from the server certificate
SSLCGI=Apache_mod_SSL_client	provides variables representing X509 V3 extensions from the client certificate
SSLCGI=Purveyor	provides Purveyor style variables

- **[NO]STMLF -** Specify files to be automatically converted to Stream-LF format. The default is to ignore conversion. STMLF allows selected paths to be converted.
- **THROTTLE=<*parameter*> -** Controls the concurrent number of scripts being processed on the path.

  See Section 4.5.

Rule
THROTTLE=*n[/u][,n,n,n,hh:mm:ss,hh:mm:ss]*
THROTTLE=FROM=<*n*>
THROTTLE=USER=<*u*>
THROTTLE=TO=<*n*>
THROTTLE=RESUME=<*n*>
THROTTLE=BUSY=<*n*>
THROTTLE=TIMEOUT=QUEUE=<*hh:mm:ss*>

Rule
THROTTLE=TIMEOUT=BUSY=<*hh:mm:ss*>

- **TIMEOUT=<*parameter*> -** Sets the appropriate timeout period on a per-path basis. The string "none" can be used to specify *no timeout*.

  These parallel the respective configuration timeout periods. See Section 8.2.

Rule	Description
TIMEOUT=<*hh:mm:ss*>, <*hh:mm:ss*>,<*hh:mm:ss*>	Keep-alive, then no-progress, then output timeouts.
TIMEOUT=KEEPALIVE= <*hh:mm:ss*>	Keep idle network connections alive for this long.
TIMEOUT=NOPROGRESS= <*hh:mm:ss*>	Terminate connection when no data is transferred to the client for this period.
TIMEOUT=OUTPUT= <*hh:mm:ss*>	Terminate connection after this period when no response data has been sent.
NOTIMEOUT	No timeouts are applied to the request.

- **WEBDAV=<*parameter*> -** Controls aspects of WebDAV processing or behaviour.

Rule	Description
WEBDAV=[NO]HIDDEN	list (default) or hide U*x *hidden* files (i.e. those with names beginning with period)
WEBDAV=[NO]LOCK	allow/apply WebDAV locking to this path
WEBDAV=[NO]PROFILE	WebDAV access according to SYSUAF profile
WEBDAV=[NO]PROP	allow/apply WebDAV 'dead' property(ies) to this path
WEBDAV=[NO]PUT=LOCK	a resource must be locked before a PUT is allowed
WEBDAV=[NO]READ	WebDAV methods allowed read this tree
WEBDAV=[NO]SERVER	WebDAV access as server account (best effort)
WEBDAV=[NO]WINPROP	when NOWINPROP windows properties are ignored and emulated
WEBDAV=[NO]WRITE	WebDAV methods allowed write to this path (implied read)
WEBDAV=LOCK=TIMEOUT=DEFAULT=	hh:mm:ss
WEBDAV=LOCK=TIMEOUT=MAX=	hh:mm:ss
WEBDAV=META=DIR=	per-path equivalent of global [WebDAVmetaDir]

- **WEBSOCKET=<*parameter*> -** Controls aspects of WebSocket processing or behaviour.

Rule	Description
WEBSOCKET=INPUT=*integer*	Specifies the size of the WEBSOCKET_INPUT mailbox buffer; in bytes.
WEBSOCKET=OUTPUT=*integer*	Specifies the size of the WEBSOCKET_OUTPUT mailbox buffer; in bytes.

Of course, as with all mapping rules, paths containing file types (extensions) may be specified so it is quite easy to apply settings to particular groups of files. Multiple settings may be made against the one path, merely separate set directives from each other with white-space. If a setting string is required to contain white-space enclose the string with single or double quotes, or curly brackets. The following example gives a small selection of potential uses.

```
# examples of SET rule usage
# -------------------------
# disable caching for selected paths
set /wasd_root/src/* NOcache
set /sys$common/* NOcache
# enable stream-LF conversion in selected directory trees
set /web/* stmlf
set /wasd_root/* stmlf
# respond with Cyrillic character set(s) from relevant directories
set /*/8859-5/* charset=ISO-8859-5
set /*/koi8-r/* charset=KOI8-R
# the Sun Java tutorial when UNZIPped contains underscores for invalid characters
set /vms/java/tutorial/* RMSchar=_
# if a request has "/plain-text/" in its path then ALWAYS return as plain-text!
set /*/plain-text/* content=text/plain
map /*/plain-text/* /*/*
# same for "/binary/"
set /*/binary/* content=text/plain
map /*/binary/* /*/*
# indicate extended file specifications on this path
set /Documents/* ODS=5
pass /Documents/* /ods5_device/Documents/*
# throttle this script's execution, 5 executing, unlimited waiting
set /cgi-bin/big_script* throttle=5
# disable server script search for this RTE
set /onerte/*  script=nofind
exec /onerte/* (CGI-BIN:[000000]ONERTE.EXE)/wasd_root/src/one/*
```

Postfix SET Rule

Path SETings may appended to any rule that contains both a template and result. This makes it possible to apply path SETings using matching final rules. For example a matching PASS rule does not require a separate, preceding SET rule containing the same path to also apply required SETings. This is more efficient (requiring less pattern matching) and tends to make the rule set less cluttered.

```
# examples of postfix SET rule usage
# ---------------------------------
# if a request has "/plain-text/" in its path then ALWAYS return as plain-text!
map /*/plain-text/* /*/* content=text/plain
# same for "/binary/"
map /*/binary/* /*/* content=text/plain
# indicate extended file specifications on this path
pass /Documents/* /ods5_device/Documents/* ODS=5
# throttle this script's execution, 5 executing, unlimited waiting
script /big_script* /cgi-bin/big_script* throttle=5
```

12.6 Reverse Mapping

Path mapping is required to get from web-space into file-space, and that mapping is not *necessarily* one-to-one. That is, /web/doc/ may not be WEB:[DOC] but for example, DKA0:[WEB.DOC] so that mapping would be

```
pass /web/* /dka0/web/*
```

Mapping paths in reverse is needed to get something like DKA0:[WEB.DOC]THIS.TXT (that may come from a $SEARCH result) back into the web-space of /web/doc/this.txt. So WASD needs paths that may be mapped using the *result* back to the *template*. In simple mappings the one rule can serve both purposes. In some situations explicit, extra rules are needed.

The above example is trivial, and if WASD needs to turn something like DKA0:[DOC]THIS.TXT into a web-space representation (URI) it makes the file-space specification into URI syntax (i.e. /dka0/web/doc/this.txt) and then scans the rules comparing that to *result* strings in the MAP rules. When one matches, the *template* component is used to generate a web-space representation - the reverse of what was done when the request was initially being processed.

The non-trivial example is often associated with concealed, search-list devices. For example, the somewhat contrived

```
$ DEFINE /SYSTEM /TRANSLATION=CONCEALED WEB DKA100:[WEB1.],DKA200:[WEB2.]
```

with which the mapping from web- to file-space can be

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

using the logical device, and quite naturally maps into file-space. WASD's file-system actions are complex and low-level, often needing to access to the underlying device (and so tend to $PARSE NOCONCEAL). Results from the above mapping can come back DKA100:[WEB1]THIS.TXT and DKA200:[WEB2]THAT.TXT and so the above mapping can't be used to get back into web-space because there is no *template* with a matchable rule.

In such a case there is a need to add explicit reverse-mapping rules (often immediately following the forward mapping rule for convenience of grouping, but rules are also a little position sensitive so some skill is required) for the purpose of getting the underlying file specifications into a form for web consumption. In the above scenario an example would be

```
pass /web/* /web/*
pass /web/* /dka100/web1/*
pass /web/* /dka200/web2/*
```

where the latter two are never hit during forward mapping (because the first rule will always map a request URI beginning /web/...) but will be hit during reverse-mapping. If a reverse

mapping exhausts the rules before finding a match the NO:[REVERSE.MAPPING.FOR.THIS]FILE.PATH! mapping is explicitly generated.

It is not always straight-forward and sometimes a decision is necessary about how the web-space is to be presented to the clients. For instance, while you easily can have multiple web-space views of the one file-space area, it is less straight-forward to have multiple web-space reverse mappings of the one file-space (as normally only the first matching rule will ever be reverse-mapped).

12.7 Mapping Examples

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

online Web link

Example of Map Rule

The *result* string of these rules may or may not correspond to to a VMS physical file system path. Either way the resulting rule is further processed before passing or failing.

1. The following example shows a path "/web/unix/shells/c" being mapped to "/web/software/unix/scripts/c", with this being used to process further rules.

```
map /web/unix/* /web/software/unix/*
```

Examples of Pass Rule

1. This example shows a path "/web/rts/home.html" being mapped to "/user$rts/web/home.html", and this returned as the mapped path.

```
pass /web/rts/* /user$rts/web/*
```

2. This maps a path "/icon/bhts/dir.gif" to "/web/icon/bhts/dir.gif", and this returned as the mapped path.

```
pass /icon/bhts/* /web/icon/bhts/*
```

3. This example illustrates HTTP status code mapping. Each of these does basically the same thing, just using one of the three possible delimiters according to the characters required in the message. The server generates a 403 response with has as its 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 "/wasd_root/script/" must be the URL format equivalent of the physical VMS specification for the directory locating the script DCL procedure. The "/web/example.conf" that followed the "/htbin/ismap" in the original URL becomes the translated path for the script.

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

2. If a URL path "/pl-bin/example/this/directory/and-file.txt" is being mapped the script name and filename become "/pl-bin/example" and "WASD_ROOT:[SRC.PERL]EXAMPLE.PL" respectively, the path information and translated become "/this/directory/and-file.txt" and "THIS:[DIRECTORY]AND-FILE.TXT", and the interpreter (run-time environment) activated to interpret the script is CGI-BIN:[000000]PERLRTE.EXE.

```
exec /pl-bin/* (cgi-bin:[000000]perlrte.exe)/wasd_root/src/perl/*
```

3. If a URL path "/conan/web/example.hlb" is being mapped the "/wasd_root/script/conan" must be the URL format equivalent of the physical VMS specification for the DCL procedure. The "/web/example.hlb" that followed the "/conan/" in the original URL becomes the translated path for the script.

```
script /conan* /wasd_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/*
```

12.8 Virtual Servers

As described in Section 4.3, virtual service syntax may be used with mapping rules to selectively apply rules to one specific service. This example provides the essentials of using this syntax. Note that service-specific and service-common rules may be mixed in any order allowing common mappings (e.g. for scripting) to be shared.

```
# a mapping rule example of virtual servers
[[alpha.domain.name:80]]
# ALPHA is the only service allowing access to VMS help directory
pass /sys$common/syshlp/*
[[beta.domain.name:80]]
# good stuff is only available from BETA
pass /good-stuff/*
# BETA has its own error report format, the others share one
pass /errorreport /httpd/-/errorreportalpha.shtml
[[gamma.domain.name:80]]
# gamma responds with documents using the Cyrillic character set
set /* charset=ISO-8859-5
[[*]]
# common file and script mappings
exec /cgi-bin/* /cgi-bin/*
exec+ /cgiplus-bin/* /cgi-bin/*
script+ /help/* /cgiplus-bin/conan/*
pass /errorreport /httpd/-/errorreport.shtml
# now the base directories for all documents
[[alpha.domain.name:80]]
/* /web/alpha/*
[[beta.domain.name:80]]
/* /web/beta/*
[[gamma.domain.name:80]]
/* /web/gamma/*
[[*]]
# catch-all rule (just in case :-)
pass /* /web/*
```

The Server Administration page WATCH report provides the capability to view the rule databse as well as rule mapping during actual request processing, using the WATCH facility.

12.9 Conditional Mapping

Deprecated and Discouraged

See Chapter 7 for current funtionality.

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

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

12.10 Mapping User Directories (*tilde* character (“~”))

The convention for specifying user web areas is “/~username/”. The basic idea is that the user’s web-available file-space is mapped into the request in place of the tilde and username.

12.10.1 Using The SYSUAF

The USER rule maps a VMS user account default device and directory (i.e. *home* directory) into a request path (Section 12.5.3). That is, the base location for the request is obtained from the VMS systems SYSUAF file. A user's home directory information is cached, to reduce load on the authorization databases. As this information is usually quite static there is no timeout period on such information (although it may be flushed to make room for other user's). Cache contents is include in the Mapping Rules Report and is implicitly flushed when the server's rules are reloaded.

The following is a typical usage of the rule.

```
USER  /~*/*  /*/www/*
```

Note the "/www" subdirectory component. It is **stongly recommended** that users never be mapped into their top-level, but into a web-specific subdirectory. This effectively "sandboxes" Web access to that subdirectory hierarchy, allowing the user privacy elsewhere in the home area.

To accomodate request user paths that do not incorporate a trailing delimiter after the username the following redirect may be used to cause the browser to re-request with a more appropriate path (make sure it follows the USER rule).

```
REDIRECT  /~*  ///~*/
```

WASD also "reverse maps" VMS specifications into paths and so requires additional rules to provide these mappings. (Reverse mapping is required during directory listings and error reporting.) For the continuing example the following rules would be required (and in the stated order).

```
USER  /~*/*  /*/www/*
REDIRECT  /~*  ///~*/
PASS  /~*/*  /user$disk/*/www/*
```

Where user home directories are spread over multiple devices (physical or concealed logical) a reverse-mapping rule would be required for each. Consider the following situation, where user directories are distributed across these devices (concealed logicals)

```
USER$GROUP1:
USER$GROUP2:
USER$GROUP2:
USER$OTHER:
```

This would require the following mapping rules (in the stated order).

```
USER  /~*/*  /*/www/
PASS  /~*/*  /user$group1/*/www/*
PASS  /~*/*  /user$group2/*/www/*
PASS  /~*/*  /user$group3/*/www/*
PASS  /~*/*  /user$other/*/www/*
```

Accounts with a search list as a default device (e.g. SYS$SYSROOT) present particular complications in this schema and should be avoided.

Note

Accounts that possess SYSPRV, are CAPTIVE, have been DISUSERED or that have expired passwords will not be mapped. A "directory not found" error report is returned.

This error was chosen to make it to make more difficult to *probe* the authorization environment, determining whether accounts exist or not.

Of course vanilla mapping rules may be used to provide for special cases. For instance, if there is requirement for a particular, privileged account to have a user mapping that could be provided as in the following (rather exagerated) example.

```
PASS  /~system/*  /sys$common/sysmgr/www/*
USER  /~*/*  /*/www/
PASS  /~*/*  /user$disk/*/www/*
```

User Account Scripting

In some situations it may be desirable to allow the average Web user to experiment with or implement scripts. With WASD 7.1 and later, and VMS V6.2 and later, this is possible. Detached scripting must be enabled, the /PERSONA startup qualifier used, and appropriate mapping rules in place. If the SET "script=as=" mapping rule specifies a tilde character then for a user request the mapped SYSUAF username is substituted.

The following example shows the essentials of setting up a user environment where access to a subdirectory in the user's home directory, [.WWW] with script's located in a subdirectory of that, [.WWW.CGI-BIN].

```
UXEC  /~*/cgi-bin/*  /*/www/cgi-bin/*  script=as=~
USER  /~*/*  /*/www/*
REDIRECT  /~*  /~*/
PASS  /~*/*  /dka0/users/*/*
```

For more detailed information see the "Scripting Overview, Introduction".

12.10.2 Without Using The SYSUAF

Deprecated and Discouraged

See Section 12.10 for current funtionality.

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

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

12.11 Cross Origin Resource Sharing

Cross-site HTTP requests are HTTP requests for resources from a domain different to the domain of the resource making the request. For instance, a resource loaded from domain one (http://domain.example) such as an HTML web page, makes a request for a resource on domain two (http://domain.foo), such as an image, using the img element (http://domain.foo/image.jpg). This occurs very commonly on the web today. Pages load a number of resources in a cross-site manner, including CSS stylesheets, images and scripts, and other resources.

Cross-site HTTP requests initiated from within browser-based applications have been subject to well-known restrictions, for well-understood security reasons. In particular, this meant that an actively processing web application could only make HTTP requests to the domain it was loaded from, and not to other domains. Developers expressed the desire to safely evolve capabilities to make cross-site requests, for better, safer web applications. The Web Applications Working Group within the W3C has recommended the new Cross-Origin Resource Sharing (CORS) mechanism, which provides a way for web servers to support cross-site access controls, which enable secure cross-site data transfers.

Basic References

This section is not a CORS reference, just the WASD implementation. Readers are referred to more authoritative CORS resources.

http://www.w3.org/TR/cors/
http://www.html5rocks.com/en/tutorials/cors/
http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
http://developer.mozilla.org/en-US/docs/HTTP/Access_control_CORS

WASD CORS

WASD supports CORS using mapping rules. This means cross-origin requests are evaluated prior to accessing any resources or activating any scripts, etc. If the request has an "Origin: .." header and the path has been *set cors=origin=..* the server performs preflighted and request checks. If CORS authorised adds CORS response headers. If not CORS authorised adds nothing. Some significant understanding of the purpose and operation of CORS is required to tailor the provision of the required response headers.

Rule	Description
CORS=AGE=*integer seconds*	Access-Control-Max-Age: response header
CORS=CRED=*true* | *false*	Access-Control-Allow-Credentials: response header
CORS=EXPOSE=*header[,header2,header3]*	Access-Control-Expose-Headers: response header
CORS=HEADERS=	Access-Control-Allow-Headers: response header
CORS=METHODS=*method[,method2,method3]*	Access-Control-Allow-Methods: response header
CORS=ORIGIN=*URL* | *	Access-Control-Allow-Origin: response header

WASD CORS Examples

1.

For a request containing

```
OPTIONS /resources/post-here/ HTTP/1.1
Host: bar.other
 . . .
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER
```

with the mapping rules

```
SET /resources/post-here/* \
    CORS=origin=* CORS=methods=POST,GET,OPTIONS \
    CORS=headers=X-PINGOTHER CORS=age=3600
```

would produce a response

```
HTTP/1.1 200 OK
 . . .
Content-Length: 0
Connection: Keep-Alive
Content-Type: text/plain
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 3600
```

2.

For a request containing

```
GET /resources/credentials/ HTTP/1.1
Host: bar.other
 . . .
Connection: keep-alive
Referer: http://foo.example/examples/credential.html
Origin: http://foo.example
```

with the mapping rules

```
SET /resources/credentials/* \
    CORS=origin=http://foo.example CORS=credEntials=true
```

would produce a response

```
HTTP/1.1 200 OK
 . . .
Content-Length: 106
Connection: Keep-Alive
Content-Type: text/plain
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Credentials: true
 . . .
```

Chapter 13

Authorization Configuration (Basics)

WASD offers a comprehensive and versatile authentication and authorization environment. A little too comprehensive, often leaving the new administrator wondering where to begin. The role of this chapter is to provide a starting place, especially for sources of authentication, along with some basic configurations. "WASD Web Services - Features and Facilities" contains a detailed explanation of all aspects. All examples here assume a standard installation and environment.

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

Changes to the authorization configuration file can be validated at the command-line before reload or restart. This detects and reports any syntactical and configuration errors but of course cannot check the *intent* of the rules.

```
$ HTTPD /DO=AUTH=CHECK
```

If additional server startup qualifiers are required to enable specific authorization features then these must also be provided when checking. For example:

```
$ HTTPD /DO=AUTH=CHECK /SYSUAF /PROFILE
```

A server's currently loaded authorization rules may also be interrogated from the Server Administration menu (see "WASD Web Services - Features and Facilities").

13.1 SYSUAF/Identifier Authentication

This setup allows any active account to authenticate using the local VMS username and password. By default not every account may authenticate this way, only those holding specified VMS rights identifiers. The examples provided in this section allows access to the WASD online Server Administration facility, and so may be followed specifically for that purpose, as well as serve as a general guide.

- Define the following logical before calling the server startup procedure. To make such a definition permanent add it to the system or Web environment startup procedures. This logical contains a startup qualifier that configures the server to allow authentication

from the SYSUAF, using VMS rights identifiers (“WASD Web Services - Features and Facilities” .

```
$ DEFINE /SYSTEM WASD_STARTUP_SERVER "/SYSUAF=ID"
$ @device:[WASD_ROOT.LOCAL]STARTUP.COM
```

After a change to a command-line qualifier of the server such as the above it needs to be restarted using the following directive.

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

- Decide on an identifier name. This can be an existing identifier, or one created for the purpose. For this example the identifier will be “WASD_WEBADMIN”. Any identifier can be created using actions similar to the following example.

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD /IDENTIFIER WASD_WEBADMIN
```

- Modify the authorization configuration file, accessed by the server using the system logical WASD_CONFIG_AUTH, to contain the following. This allows full access to the online Server Administration facility and [.LOCAL] directory (and no world access). Additional paths may be added as required, and of course multiple identifiers may be created and used for multiple realms and paths.

```
["Web Admin"=WASD_WEBADMIN=id]
/httpd/-/admin/* r+w
/wasd_root/local/* r+w
```

- The identifier must then be granted to those accounts allowed to authenticate in this way.

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> GRANT /IDENTIFIER WASD_WEBADMIN SYSTEM
```

- Using this approach useful discrimination may be exercised. For instance, one identifier for Web administrators, another (or others) for different authentication requirements.

```
["Web Admin"=WASD_WEBADMIN=id]
/wasd_root/local/* r+w
/httpd/-/admin/* r+w
["Area Access"=area-identifier-name=id]
/web/area/* r+w ; r
```

Of course the one account may hold multiple identifiers and so may have access to various areas.

```
UAF> GRANT /IDENTIFIER WASD_WEBADMIN SYSTEM
UAF> GRANT /IDENTIFIER area-identifier-name SYSTEM
```

Using VMS rights identifiers allows significant granularity in providing access.

After Changes

If the WASD_CONFIG_AUTH configuration file is changed, or rights identifiers are granted or revoked from accounts, the server should be directed to reload the file and purge any cached authorization information.

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

13.2 Other Authentication

Other sources of authentication are available, either by themselves or used in the same configuration file (different realms and paths) as those already discussed “WASD Web Services - Features and Facilities” . Non-SYSUAF sources do not require any startup qualifier to be enabled.

- **ACME** DOIs (Authentication and Credential Management Extension, Domains of Interpretation) may be used to authenticate requests.

  ```
  ["Whatever you want to call it!"=doi=ACME]
  /web/area/* r+w
  ```

- **Simple lists** contain usernames and unencrypted passwords. These are plain-text files, created and modified using any desired editor.

  ```
  ["Whatever you want to call it!"=list-name=list]
  /web/area/* r+w
  ```

  This is a very simple arrangement, with little inherent security. Lists are more useful when grouping names together for specifying which group may do what to where.

- **HTA databases** are WASD-specific, binary repositories of usernames, encrypted passwords, capabilities, user and other detail.

  ```
  ["Whatever you want to call it!"=HTA-database-name=HTA]
  /web/area/* r+w
  ```

  These databases may be administered using the online Server Administration facility (“WASD Web Services - Features and Facilities” or the HTAdmin command-line utility (“WASD Web Services - Features and Facilities” , are quite secure and versatile.

- **External agents** are authentication and authorization scripts executed on demand, under the control-of but external to the server. It is possible for a site to write its own, custom authorization agent.

  ```
  ["Whatever you want to call it!"=agent-name=agent]
  /web/area/* r+w
  ```

  Two variations on a versatile LDAP authenticator and a CEL-compatible authenticator, along with example code is available in the WASD_ROOT:[SRC.AGENT] directory.

- **X.509** establishes identity based on Public Key Infrastructure (PKI) authentication certificates. This is only available for SSL transactions.

  ```
  [X509]
  /web/area/* r+w
  ```

- **RFC1413** IETF document describes an identification protocol that can be used as a form of *authentication* within this realm.

```
["Whatever you want to call it!"=RFC1413;A_PROJECT=list]
/web/area/* r+w ; r
```

13.3 Read and Write Groupings

WASD allows separate sources for groups of usernames to control read and write access in a particular realm (“WASD Web Services - Features and Facilities” . These groups may be provided via simple lists, VMS identifiers, HTA databases and authorization agents. The following example shows an identifier authenticated realm with full and read-only access controlled by two simple lists. For the first path the world has no access, for the second read-only access (with the read-only grouping becoming basically redundant information).

```
["Realm Name"=identifier_name=id;full_access_name=list;read-only_name=list]
/web/area/* r+w ;
/web/another-area/* r+w ; r
```

13.4 Considerations

Multiple authentication sources (realms) may be configured in the one WASD_CONFIG_ AUTH file.

Multiple paths may be mapped against a single authentication source.

Any path may be mapped only once (for any single virtual service).

Paths may have additional access restrictions placed on them, including client host name, username, etc. (“WASD Web Services - Features and Facilities” .

The configuration file is loaded and stored by the server at startup. If changed it must be reloaded to take effect. This can be done manually using

```
$ HTTPD/DO=AUTH=LOAD
```

Authentication information is cached. Access subsequently removed or modified will not take effect until the entry expires, or is manually purged using

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

Failed attempts to authenticate against a particular source are limited. When this is exceeded access is always denied. If this has happened the cache must be manually purged before a user can successfully authenticate

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