

WASD Hypertext Services - Environment Overview

November 2009

For version 10.0 release of the WASD VMS Web Services.

Abstract

This document is a guide to supporting Web documents within the WASD Web Services environment. It is **not** a tutorial on writing HTML documents.

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

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"

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

Defence Science and Technology Organisation

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

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

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

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

+61 (8) 82596189 (bus)

+61 (8) 82596673 (fax)

PO Box 1500
Edinburgh
South Australia 5111

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.ENV]WASD_ENV.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 Hypertext Services

Copyright © 1996-2008 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.

HT_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 ...

OpenSSL Project

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

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

Eric A. Young

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

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

Free Software Foundation

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

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

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.

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 acknowledgement of the developers of this program.

RSA Data Security

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

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

Bailey Brown Jr.

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

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

Other

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

MultiNet is a registered trademark of Process Software Corporation.

Pathway is a registered trademark of Attachmate, Inc.

TCPware is a registered trademark of Process Software Corporation.

Ghostscript is Copyright (C) 2005 artofcode LLC, Benicia, CA. All rights reserved.

Contents

Chapter 1 Introduction

Chapter 2 Document Access and Specification

2.1	Document Content Type	2-1
2.2	Explicitly Specifying Content-Type	2-2
2.3	Document Specification	2-3
2.3.1	Absolute File Path	2-3
2.3.2	Partial (or Relative) File Path	2-3
2.4	Extended File Specifications (ODS-5)	2-4
2.4.1	Characters In Request Paths	2-4
2.4.2	Characters In Server-Generated Paths	2-5
2.4.3	Document Cache	2-5

Chapter 3 Directory Listing

3.1	Controlling Access To A Directory	3-2
3.2	“Hidden” Files	3-2
3.3	Server Directives	3-3
3.3.1	Layout	3-3
3.3.2	Readme Files	3-4
3.3.3	Listing Delimiters	3-4
3.3.4	Suppressing Directories	3-5
3.3.5	Listing Refresh and Expiry	3-5
3.3.6	Scripting From Directory Listings	3-5
3.3.7	Auto-Scripting	3-5
3.3.8	Specifying Content-Type	3-6
3.4	Directory Tree	3-6

Chapter 4 Server Side Includes (SSI)

4.1	Virtual Documents	4-2
4.2	Last-Modified Information	4-4
4.3	Pre-Expiring Documents	4-4
4.4	Directive Syntax	4-5
4.5	Directives	4-5
4.5.1	#ACCESSES	4-6
4.5.2	#CONFIG	4-7
4.5.3	#DIR	4-8
4.5.4	#DCL	4-8
4.5.5	#ECHO	4-9
4.5.6	#ELIF	4-10
4.5.7	#ELSE	4-10
4.5.8	#ENDIF	4-10
4.5.9	#EXEC	4-11
4.5.10	#EXIT	4-11
4.5.11	#FCREATED	4-11
4.5.12	#FLASTMOD	4-11
4.5.13	#FSIZE	4-12
4.5.14	#IF	4-12
4.5.15	#INCLUDE	4-13
4.5.16	#MODIFIED	4-13
4.5.17	#ORIF	4-14
4.5.18	#PRINTENV	4-14
4.5.19	#SET	4-14
4.5.20	#SSI	4-15
4.5.21	#STOP	4-15
4.6	Variables	4-15
4.7	Flow Control	4-17
4.8	Query Strings	4-18
4.9	File and Virtual Specifications	4-19
4.9.1	THE_FILE_NAME	4-19
4.10	Time Format	4-19
4.11	OSU Compatibility	4-22
4.12	Script-Generated SSI Documents	4-23

Chapter 5 Clickable Image Support

5.1	Image Configuration File	5-2
5.2	Examples	5-3

Chapter 6 Document Searching

6.1	Plain-Text Search	6-1
6.2	HTML Search	6-1
6.3	Search Syntax	6-2
6.3.1	“ISINDEX” Search	6-2
6.3.2	Standard Search Form	6-3
6.3.3	Forms-Based Search	6-3
6.3.4	Search Options	6-3
6.3.5	Example Search Form	6-4

Chapter 7 VMS Help and Text Libraries

Chapter 8 Bookreader Books and Libraries

Chapter 9 Web Document **Update**

Chapter 1

Introduction

The document **assumes** a basic understanding of hypertext technologies and uses some terms without explaining them (e.g. HTTP, HTML, URL, CGI, etc.) It is **not** a tutorial on writing HTML documents. The reader is referred to documents specifically on these topics (some are available online within WASD, but are often best consulted from the Internet WWW).

WASD is using hypertext technologies (a.k.a. Web and WWW) to assist in providing an integrated and commonly-accessible information environment. The VMS implementation of this has gone to considerable lengths to integrate as much as possible, all forms of VMS information. This includes plain-text documents such as programming sources and environment/include files, release notes, etc., as well as non-plain-text information like HELP, Bookreader, etc.

The purpose of this document is to describe some of the facilities within the WASD VMS hypertext environment that can be used from within WASD HTML documents. The WASD VMS hypertext infrastructure conforms to all the basic conventions of Web technology, although having some facilities specific-to, or tailored-for its VMS environment.

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

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

Chapter 2

Document Access and Specification

Arbitrary documents may not be accessed.

The server can only access files where the path is allowed according to a specified set of rules specified within the hypertext environment.

Documents must be read-accessible.

The server can only access files that are world readable, or that have an ACL specifically controlling access for “HTTP\$SERVER”, the server account.

2.1 Document Content Type

Document (file) retrieval is initiated by providing the server with the file specification as a URL path. Server configuration determines the format in which the file is returned to the client. It may contain text or images immediately displayable by the browser, or by a viewer external to the browser may be spawned. The server may automatically activate a script to provide a gateway to non-native information (see description of [AddType] configuration directive in the Technical Overview). The file type (extension) determines the content type by which the server returns (and/or interprets) the file.

The following table lists some of the current file types (as examples) and their associated MIME-style content type. HTML documents are presented layed-up according to the full HTML-capabilities of the browser. Plain-text documents are presented in a fixed-font format. Other types require an external viewer to be activated. Here are a few examples.

.BKB	Bookreader document (BNU)	text/html, gateway script activated
.BKS	Bookreader shelf (BNU)	text/html, gateway script activated
.C	C source	text/plain
.COM	DCL procedure	text/plain
.CONF	configuration file	text/plain
.CPP	C++ source	text/plain
.DECW\$BOOK	Bookreader document	text/html, gateway script activated
.FOR	Fortran source	text/plain
.GIF	GIF image	image/gif
.H	C header	text/plain
.HLB	VMS Help library	text/html, gateway script activated
.HTML	HyperText Markup Language	text/html
.HTM	HyperText Markup Language	text/html
.JPG	JPEG image	image/jpeg
.LIS	Listing	text/plain
.MAR	Macro source	text/plain
.PAS	Pascal source	text/plain
.PRO	IDL source	text/plain
.PS	PostScript	application/PostScript
.TEXT	Text	text/plain
.TLB	VMS text library	text/html, gateway script activated
.TXT	Text	text/plain
.SHTML	HyperText Markup Language	pre-processed text/html
.ZIP	zipped file	application/binary

If other file types are required to be defined contact the Web administrator.

2.2 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 also possible to “force” the content-type for all files in a particular directory. See Section 3.3.8.

2.3 Document Specification

For the “http:” protocol, file and directory locations are specified using URL path syntax where slash-separated (“/”) elements delineate a hierarchy leading to a data item. Anyone familiar with the syntax of the Unix file system, or the MS-DOS file system (where backslashes are hierarchy delimiters), will feel at home with URL syntax. Specifications under VMS are not case-sensitive.

A VMS directory specification

```
WEB:[TECHNICAL.HTML-PRIMER]
```

would be represented in URL syntax as

```
/web/technical/html-primer/
```

and a VMS file specification

```
WEB:[TECHNICAL.HTML-PRIMER]HTML-PRIMER.HTML
```

represented as

```
/web/technical/html-primer/html-primer.html
```

Note

It is not required (although not forbidden) to supply a VMS *master file directory* component (“[000000]”, “[000000].”, etc.) in a URL specification. Hence the file specification

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

should be represented as

```
/web/home.html
```

2.3.1 Absolute File Path

A file may be specified using an *absolute*, or full path. This must specify the location of the file exactly. Absolute paths **always** begin with a forward-slash (“/”). For example:

```
/web/committee/minutes/1994/1994-09-27.txt  
/web/committee/constitution.txt  
/web/committee/membership/fred-bloggs.txt
```

2.3.2 Partial (or Relative) File Path

(Strictly speaking, it is a function of the client to construct a full URL from such a relative URL before sending the request to the server.)

A file may be specified *relative* to its current location. That is, a current document (or menu) may specify another document file relative to itself. This may be at the current level, a subdirectory, or in another part of the directory tree related to the current. Relative paths **never** begin with forward-slash (“/”).

For example, documents at the same level as the current may be specified without any hierarchy being indicated:

```
1994-07-22.txt
1994-08-24.txt
1994-09-27.txt
```

Documents at an inferior point in the hierarchy may be specified as in the following example:

```
1993/1993-02-17.txt
1993/reports/membership.txt
other/etc.txt
```

Documents in a related part of the hierarchy may be referenced using the “../” construct. As with MS-DOS and Unix this syntax indicates the immediately superior directory.

```
../other_committee/1993/1993-02-17.txt
../other_committee/1993/reports/balance-sheet.txt
../../other_section/committee/constitution.txt
```

2.4 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.

2.4.1 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.

The WASD server will accept request paths for file specifications in both formats, URL-encoded and RMS-escaped. Of course characters absolutely forbidden in request paths must still be URL-encoded, the most obvious example is the space. RMS will accept the file name “This^ and^ that.txt” (i.e. containing escaped spaces) but the request path would need to be specified as “This%20and%20that.txt”, or possibly “This^%20and^%20that.txt” although the RMS escape character is basically redundant.

Unlike for ODS-2 volumes, ODS-5 volumes do not have “invalid” characters, so unlike with ODS-2 no processing is performed by the server to ensure RMS compliance.

2.4.2 Characters In Server-Generated Paths

When the server generates a path to be returned to the browser, either in a viewable page such as a directory listing or error message, or as a part of the HTTP transaction such as a redirection, the path will contain the URL-encoded equivalent of the *canonical form* of an extended file specification escaped character. For example, the file name “This^_and^_that.txt” will be represented by “This%20and%20that.txt”.

When presenting a file name in a viewable page the general rule is to also provide this URL-equivalent of the unescaped file name, with a small number of exceptions. The first is a directory listing where VMS format has been requested by including a version component in the request file specification. The second is in similar fashion, but with the *tree* facility, displaying a directory tree. The third is in the navigation page of the *UPDate* menu. In all of the instances the canonical form of the extended file specification is presented (although any actual reference to the file is URL-encoded as described above).

2.4.3 Document Cache

The Web server is most commonly set up to cache static documents (files). A cache is higher speed storage, in-memory, in the server itself. Cached documents are checked periodically for changes when being requested. Changes to a file are determined by the comparing the modification date/time and file length. A common check period is one minute, though it can set longer or even disabled. If a document has changed the old one is discarded from cache (called invalidation) and the new one loaded into cache while being transfered to the client.

After making changes to a document it is possible the server will continue to serve the old one for a short period. This can be overridden by using the browser’s *Reload* facility. This directs the server to go and check the on-disk file regardless, invalidating it if necessary.

Chapter 3

Directory Listing

A directory listing is sometimes referred to as a document *Index*, and is generally titled “Index of . . .”.

Unless disabled by the server’s configuration, a directory listing is recognised by the server whenever a wildcard is present in a specification and there is no query string directing another activity (e.g. a document search). Compliant with other hypertext implementations, a directory listing is also generated if a URL specifies a directory only and that directory contains no home page.

All specifications must be made using URL-style paths. See Section 2.3.

The directory listing is designed to look very much like the basic layout of other servers, except that all directories are grouped at the top. In the opinion of the author, this looks and functions better than when interspersed with the files, as is otherwise common. The default listing provides:

- iconic indication of the data type
- file name
- last revision date/time
- size
- description

The description can be either be just that, a description of the role of that type of file under VMS, or if presented within quotes, an HTML document’s own internal description taken from the “<TITLE></TITLE>” element.

Note that directory listings only processes the physical file system. This may or may not correspond to the hypertext environment’s virtual mappings.

The following link illustrates the directory listing format:

```
<A HREF=" * . * "> * . * </A>
```

[online demonstration](#)

VMS-ish Format

The default listing has a *generic* WWW look about it, however it can be made to look a little more like the format of the VMS “DIRECTORY” command. In this mode the directories are presented as VMS subdirectories, the version number is shown, if a version wildcard was included in the specification then all matching versions are shown, the size is presented in used and allocated blocks, and automatic script activation is disabled. The VMS-style format is enabled by providing an explicit or wildcard version number with the specification, as in the following example: `*.*;.` [online demonstration](#)

Listing Icons

By default (and generally) WASD installations are configured to return a binary file (usually triggering a browser “save-as” dialog) for unknown content-types. For such files and for non-text files in general, a directory listing icon becomes a link to a plain-text version of the file (regardless of the actual content). So for files containing such plain-text (often *readme* files with “interesting” file names) this becomes a convenient way of access the content.

3.1 Controlling Access To A Directory

The following files (empty, or not), when within a specific directory regulate access to that directory, and the listing of any parent directory or subdirectories.

- **.WWW_HIDDEN** - Renders the directory completely invisible to the directory listing mechanism. Files within the directory may still be accessed if specified explicitly but the directory content itself cannot be listed by any means.
- **.WWW_NOWILD** - Renders the directory incapable of being listed using “*.*” characters at the end of the path, even if allowed by the server. This is a little different to **.WWW_HIDDEN**, which hides the directory completely. The *no-wild* still allows a directory without a home page to list as a directory, it does however **prevent the forced listing using the “*.*” syntax**.
- **.WWW_NOP** - Any parent directory is not listed.
- **.WWW_NOS** - Any subdirectories are not listed.
- **.WWW_NOPS** - Any parent directory or subdirectories are not listed.

3.2 “Hidden” Files

Any file name beginning with a period is hidden from the directory listing mechanism (i.e. in VMS parlance it has only a type/suffix/extension). If specifically accessed they will be retrieved however. Hence the following files would not appear in a directory listing:

```
.WWW_NOPS
.CANT_BE_SEEN
.HIDDEN_FROM_VIEW
.;1
```


3.3 Server Directives

The WASD server behaviour can be modified using *server directives*. For directory listings this involves the inclusion of a query string beginning with “?httpd=index”. The server detects this query string and processes it internally, changing the default action of directory listings.

Multiple directives can be combined by concatenating them with intervening ampersands, as per normal URL syntax.

```
?httpd=index&autoscript=no
?httpd=index&readme=no
?httpd=index&type=text/plain
?httpd=index&layout=format
?httpd=index&script=script-name
?httpd=index&script=script-name&readme=no
?httpd=index&delimit=none&readme=no&nos=yes
```

3.3.1 Layout

Allows specification of the directory listing layout from the URL, overriding the server default. The layout directive 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 (one with colon-separated parameter) and optional leading decimal number specifying the width. When 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 (often 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 default layout is:

```
I__L__R__S__D
```

The following provide other examples:

```
?httpd=index&layout=UI__L__R__S__D
?httpd=index&layout=I__L__R__S:b__D
?httpd=index&layout=I__L__R__S__D
?httpd=index&layout=I__15L__S__D
?httpd=index&layout=15L__9R__S
?httpd=index&layout=15N_9C_9R_S
```

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 preferred to have the default display in 1000 byte kilos then set the directory listing layout using:

```
?httpd=index&layout=I__L__R__S:d__D
```

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

[online demonstration](#)

3.3.2 Readme Files

When a directory listing is generated any “README.”, “README.TXT” or “README.HTML” file (or others as configured for the particular server) in the directory will have the contents displayed immediately below the title of the page. This allows additional information on the directory’s contents, function, etc., to be presented. This can be suppressed by appending the following query-string to the directory specification, as in the accompanying example:

```
?httpd=index&readme=no
<A HREF="*.?*?httpd=index&readme=no">*. * (no <I>readme</I>s)</A>
```

Read-me files can be SSI documents if configured by the server administrator. General SSI guidelines apply to these, see Chapter 4

3.3.3 Listing Delimiters

A directory listing is normally delimited by a header, comprising an “Index of”, column headings and horizontal line, and a footer, comprising a horizontal line. This default behaviour may be modified using the “delimiter=” directive.

- **header** - a header comprising a horizontal rule and column heading is generated
- **footer** - a footer comprising a horizontal rule is generated
- **none** - no header or footer is generated
- **both** - both header and footer is generated (default)

```
?httpd=index&delimiter=none
?httpd=index&delimiter=top
```

3.3.4 Suppressing Directories

Parent and subdirectories may be suppressed in a listing using the “nop”, “nops” and “nos” directives. These parallel the purpose of the directory listing control files described in Section 3.1, and if set to true suppress the listing of the corresponding directories.

- **nop** - any parent directory is not listed
- **nops** - any subdirectories are not listed
- **nos** - any parent directory or subdirectories are not listed

```
?httpd=index&nop=yes  
?httpd=index&nops=yes  
?httpd=index&nos=yes
```

3.3.5 Listing Refresh and Expiry

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 often disabled by default. Individual directory listings may have either default behaviour over-ridden using syntax similar to the following examples:

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

3.3.6 Scripting From Directory Listings

When a directory listing is requested a script name can be specified to be used as a prefix to all of the file links in the listing. When the client selects a file link the script specified is implicitly activated.

```
?httpd=index&script=script_name  
<A HREF="*.?*?httpd=index&script=print">print *.*</A>
```

[online demonstration](#)

3.3.7 Auto-Scripting

The server’s *auto-scripting* facility (see description of [AddType] configuration directive in the Technical Overview) can be suppressed by appending the following query-string to the directory specification, as in the accompanying example:

```
?httpd=index&autoscript=no  
<A HREF="*.?*?httpd=index&autoscript=no">get me *.*</A>
```

This implies that any file accessed from the listing will be transferred without any data conversion possible due to script activation. The browser must then process the document in some fashion (often by activating a *save as* dialog).

3.3.8 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. See Section 2.2.

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.

It is possible 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](#)

3.4 Directory Tree

The "Tree" internal script allows a directory tree to be generated. This script is supplied with a directory name from which it displays all subdirectories in a hierarchical layout, showing subordinancies. Selecting any one of the subdirectories displayed generates a directory listing (see Chapter 3).

Appending a file specification (with or without wildcards) to the directory name results in the any directory listing displaying only files matching the specification. To display all files a "*" should always be appended.

Note that this script only processes the physical file system. This may or may not correspond to the hypertext environment's virtual mappings.

To enable the VMS-style directory listing format, or to use any of the directory server directives, append one, or a combination of, the following query strings to the directory specification:

```
?httpd=index&autoscript=no  
?httpd=index&readme=no  
?httpd=index&script=script-name  
?httpd=index&script=script-name&readme=no
```

```
<A HREF="/tree/wasd_root/*.*">/wasd_root/</A> tree
```

```
<A HREF="/tree/wasd_root/*.*;*">/wasd_root/ (VMS-ish)</A> tree
```

[online demonstration](#)

Note that this activity is I/O intensive, and can take a considerable period if the tree is extensive.

Note

The “tree” internal script supercededs the “Dtree” external script which has been retired.

Chapter 4

Server Side Includes (SSI)

The HTML pre-processor is used to provide dynamic information inside of an otherwise static, HTML (HyperText Markup Language) document. The HTTPd server provides this as internal functionality, scanning the input document for special pre-processor *directives*, which are replaced by dynamic information based upon the particular directive.

As of version 5.1 WASD SSI has been enhanced to provide flow-control statements, allowing blocks of the document to be conditionally processed, see Section 4.7. These extensions allow quite versatile documents to be created without resorting to script processing.

Two documents are provided as examples of SSI processing.

- **A simple SSI document.**

WASD_ROOT:[DOC.ENV]SSI.SHTML

[online demonstration](#)

- **An SSI document using variable assignment and flow-control.**

WASD_ROOT:[DOC.ENV]SSI.SHTML

[online demonstration](#)

By default the HTML pre-processor is invoked when the document file's extension is ".SHTML". As there is a significant overhead with pre-processed HTML compared to normal HTML, it should only be used when it serves a useful documentary purpose, and not just for the novelty.

Essential compatibility with OSU Server Side Includes is provided. This may ease any transition between the two. See Section 4.11 for further information.

4.1 Virtual Documents

One effective use for pre-processed HTML is the creation of single virtual documents from two or more physical documents. That is, the pre-processed document is used to include multiple physical documents, that may even be independently administered, to return a composite document to the client. This is a relatively low-overhead activity as SSI goes, but because it is a dynamic document, without some extra considerations (see Section 4.2).

Example 1

This provides an example of the efficient use of SSI processing to create virtual documents. Each page will comprise a header (containing the body tag and page header, etc), the document proper and a footer (containing the end-of-page information, modification date, and end-body tag, etc).

```
<HTML>
<HEAD>
<TITLE>Just an example!</TITLE>
</HEAD>
<!--#include virtual="header.shtml" -->
<P> This is the document information.
<P> Blah, blah, blah.
<!--#include virtual="/web/common/footer.shtml" -->
</HTML>
```

A more efficient variant places the document proper in its own, plain HTML file which is then #included (it is much, much, much more efficient for the server to *throw* a file at the network, than parse every character in one ;^)

```
<HTML>
<HEAD>
<TITLE>Just an example!</TITLE>
</HEAD>
<!--#include virtual="header.shtml" -->
<!--#include virtual="example.html" -->
<!--#include virtual="footer.shtml" -->
</HTML>
```

Example 2

This example provides a seemingly more convoluted, but very much more powerful configuration, that uses recursion to greatly simplify maintenance of common-layout documents for the end-user.

File 1; the document accessed via the browser URL, doesn't matter what its name is, this configuration is completely naming independent.

```
<!--#ssi
#if var={PARENT_FILE_NAME} eqs=""
  #set var=TITLE value="Just an Example"
  #include virtual="/web/common/template.shtml"
#else
  #include virtual="document.html"
#endif
-->
```

File 2; the TEMPLATE.SHTML referred to by the first include above.

```
<!--#ssi
#include virtual="/web/common/header.shtml"
#include virtual="{DOCUMENT_ROOT}header.html" fmt="?"
#include virtual="{DOCUMENT_URI}"
#include virtual="{DOCUMENT_ROOT}footer.html" fmt="?"
#include virtual="/web/common/footer.shtml"
-->
```

File 3; the DOCUMENT.HTML referred to by the second include in file 1.

```
<P> This is just a <B>bunch of HTML</B>!
```

This is an explanation of how it works . . .

1. the browser accesses file 1 via a URL
2. processing begins with file 1
3. file 1 checks if it has a parent (is the first file processed),
it doesn't and so . . .
4. file 1 set a variable named TITLE
5. file 1 #includes file 2, a site-common template
6. file 2 substitutes the TITLE variable contents as the document title
7. file 2 #includes a site-common header
8. file 2 #includes an optional document-local header
9. **here's the interesting bit . . .**
file 2 now re-#includes the original document, file 1 (!!)
10. file 1 checks if it has a parent (is the first file processed),
it does and so . . .
11. file 1 #includes file 3 (the actual contents of the document)
12. file 3 is a plain HTML document, just added to the output
13. file 1 is now exhausted and processing returns to file 2
14. file 2 #includes an optional document-local footer
15. file 2 #includes a site-common footer
16. file 2 is exhausted and processing returns to file 1
17. file 1 is exhausted and processing stops

The following link provides an example of such a virtual document.

[online demonstration](#)

4.2 Last-Modified Information

SSI documents generally contain dynamic elements, that is those that may change with each access to the document (e.g. current date/time). This makes evaluation of any document modification date difficult and so by default no “Last-Modified: *timestamp*” information is supplied against an SSI document. The potential efficiencies of having document timestamps, so that requests can be made for a document to be returned only if modified after a certain date/time (“If-Modified-Since: *timestamp*”), are significant against the CPU overheads of processing SSI documents.

WASD allows the document author to determine whether or not a last-modified header field should be generated for a particular document and which contributing file(s) should be used to determine it. This is done using the `#modified` directive. If a virtual document is made up of multiple source documents (files) each can be assessed using multiple `virtual=` or `file=` tags, the most recently modified will be used to determine if the virtual document has been modified, and also to generate the last-modified timestamp.

The *if-modified-since* tag compares the determined revision date/time of the document file(s) with any “If-Modified-Since:” timestamp supplied with the request. If the virtual document’s revision date/time is the same or older than the request’s then a not-modified (304 status) header is generated and sent to the client and document processing ceases. If more recent an appropriate “Last-Modified:” header field is added to the document and it continues to be processed.

If a request has a “Pragma: no-cache” field (as with Navigator’s *reload* function) the document is always generated (this is consistent with general WASD behaviour). The following example illustrates the essential features.

```
<!--#ssi
#modified
#modified virtual="/web/common/header.shtml"
#modified virtual="header.html" fmt="?"
#modified virtual="index.html" fmt="?"
#modified virtual="footer.html" fmt="?"
#modified virtual="/web/common/footer.shtml"
#modified if-modified-since
-->
```

This construct should be placed at the very beginning of the SSI document, and certainly before there is any chance of output being sent to the browser. Once output to the client has occurred there can be no change to the response header information (not unreasonably).

4.3 Pre-Expiring Documents

SSI preprocessed documents are *dynamic* in the sense that the information presented can be different every time the document is generated (e.g. if time directives are included). If it is important that each time the document is accessed it is regenerated then an HTML META tag can be included in the HTML header to cause the document to *expire*. This will result in the document being reloaded with each access. This can be accomplished two ways.

- Use the `#modified` directive to include an “Expires: *timestamp*” response header field. Place the following construct at the beginning of the SSI document.

```
<!--#modified expires="Fri, 13 Jan 1978 14:00:00 GMT" -->
```

An alternative, if the objective is to pre-expire the document, is to specify an expiry of zero. This is specially handled by the SSI engine. It adds an expiry response header field, plus cache-control header fields to suppress document caching (on compliant browsers).

```
<!--#modified expires="0" -->
```

4.4 Directive Syntax

The syntax follows closely that used by the other implementations, but some directives are tailored to the WASD and VMS environment. The directive is enclosed within an HTML comment and takes the form:

```
<!--#directive [[tag1="value"] [tag2="value"] ...] -->
```

A *tag* provides parameter information to the directive. A directive may have zero, one or more parameters. Values supplied with any tag may be literal or via variable substitution (see Section 4.6). A value must be enclosed by quotation marks if it contains white-space.

A directive **can be split over multiple lines** provided the new line begins naturally on white-space within the directive. For example, this is correctly split

```
<!--#echo  
created[="<EMPHASIS>(time-format)"] -->
```

while the following is not (and would produce an error)

```
<!--#echo creat  
ed[="<EMPHASIS>(time-format)"] -->
```

Directive and tag keywords are case insensitive. The tag value may or may not be case sensitive, depending upon the command/tag. Generally the effect of a command is to produce additional text to be inserted in the document, although it is possible to control the flow of processing in a document with decision structures.

4.5 Directives

SSI Directives

Directive	Description	Section
#accesses	document access count	Section 4.5.1
#config	document processing options	Section 4.5.2
#dir	directory listing	Section 4.5.3
#dcl	DCL command processing	Section 4.5.4
#echo	output information	Section 4.5.5
#elif	flow control	Section 4.5.6

Directive	Description	Section
#else	flow control	Section 4.5.7
#endif	flow control	Section 4.5.8
#exec	same as “#dcl”	Section 4.5.9
#exit	flow control, stop current document processing	Section 4.5.10
#fcreated	output file creation date/time	Section 4.5.11
#flastmod	output file last modification date/time	Section 4.5.12
#fsize	output file size	Section 4.5.13
#if	flow control	Section 4.5.14
#include	include a text file or another SSI document	Section 4.5.15
#modified	HTTP response control	Section 4.5.16
#orif	flow control	Section 4.5.17
#printenv	list document variables	Section 4.5.18
#set	assign value to a document variable	Section 4.5.19
#ssi	block of SSI statements	Section 4.5.20
#stop	stop SSI processing completely	Section 4.5.21

4.5.1 #ACCESSES

The *#accesses* directive allows the number of times the document has been accessed to be included. It does this by creating a counter file in the same location and using the same name with a dollar symbol appended to the type (extension). The count may be reset by deleting the file. This is an expensive function (in terms of file system activity) and so should be used appropriately. It can be disabled by server configuration. Three tags provide additional functionality:

- **ORDINAL**

```
<!--#accesses ordinal -->
```

Provides the count as 1st, 2nd, 3rd, 4th, 5th . . . 10th, 11th, 12th . . . 120th, 121st, 122nd, etc.

- **SINCE**

```
<!--#accesses since="text" -->
```

This tag includes the specified text immediately after the access count is displayed, then adds the creation date of the counter file.

- **TIMEFMT**

```
<!--#accesses since="text" timefmt="[time-format]" -->
```

Allows the time format of the *since* tag to be supplied, where *time-format* is specified according to Section 4.10.

4.5.2 #CONFIG

The *#config* directive allows time and file size formats to be specified for all subsequent directives providing these values. Optional specifications for individual directives may still be made, and override, do not supercede, any specification made using a *config* directive. A *config* directive may be made once, or any number of times in a document, and applies until another is made, or until the end of the document.

- **ERRMSG**

```
<!--#config errmsg="string" -->
```

This directive allows the error message generated if a problem problem processing the SSI document occurs (e.g. miss-spelled directive) to be specified in the document.

- **TIMEFMT**

```
<!--#config timefmt="time-format" -->
```

Where *time-format* is specified according to Section 4.10.

- **SIZEFMT**

```
<!--#config sizefmt="size-format" -->
```

Where *size-format* is specified using the following keywords:

- “abbrev” (as bytes, kbytes, Mbytes)
- “blocks” (VMS blocks, used)
- “bytes” (e.g. “1,256,731 bytes”)

- **TRACE**

```
<!--#config trace="1|0" -->
```

Switches document processing trace on or off, intended for use when debugging more complex or flow-controlled SSI documents.

Output from a trace is colour-coded.

- **Blue** - As a line is read from the document is is displayed in blue. The text is preceded by a square-bracketed source file line number and flow-control level.
- **Red** - As an SSI statement is actually processed it is displayed in red. Due to document parsing this may occur at some point after the line is read from file.
- **Magenta** - As variables are set or read the variable name and value is displayed. A variable set has the name separated from the value by an equate symbol (“=”), when being read the character is a full-colon (“:”).
- **Black** - Document (HTML and text) output is displayed as black plain text.

The following link provides an example of a document trace.

[online demonstration](#)

4.5.3 #DIR

The `#dir` directive generates an *Index of . . .* directory listing inside an HTML document. Apart from not generating a title (it is up to the pre-processed document to title, or otherwise caption, the listing) it provides all the functionality of the WASD HTTPd directory listing (see Chapter 3), including query string format control via the “par=” parameter (note that from the “?httpd=index” introducer used with directory listings is not necessary from SSI). It is an WASD HTTPd extension to pre-processed HTML.

- **FILE**

Listing specified using a VMS file path.

```
<!--#dir file="file-name" [par="server-directive(s)"] -->
```

- **VIRTUAL**

Listing specified using URL-style syntax.

```
<!--#dir virtual="path" [par="server-directive(s)"] -->
```

For example:

```
<!--#dir /wasd_root/src/httpd/" -->
```

```
<!--#dir /wasd_root/src/httpd/*.c" par="layout=UL__S&nops=yes" -->
```

4.5.4 #DCL

The `#dcl` directive executes a DCL command and incorporates the output into the processed document. It is an WASD HTTPd extension to the more common `exec` directive, which is also included.

By default, output from the DCL command has all HTML-forbidden characters (e.g. “<”, “&”) escaped before inclusion in the processed document. Thus command output cannot interfere with document markup, but nor can the DCL command provide HTML markup. This behaviour may be changed by appending the following tag to the directive:

```
type="text/html"
```

Some `#dcl` directives are for *privileged* documents only, documents defined as those being owned by the SYSTEM account, and not being world-writeable. The reason for this should be obvious. There are implicit security concerns about any document being able to execute any DCL command(s), even if it is being executed in a completely unprivileged process. Hence only *innocuous* commands are allowed in standard documents.

- **SAY**

Execute the DCL “WRITE SYS\$OUTPUT” command, using the specified parameter.

```
<!--#dcl say="hello." -->
```

- **SHOW**

Execute the DCL “SHOW” command, using the specified parameter.

```
<!--#dcl show="device/full tape1:" -->
```

- **DIR**

Execute the DCL “DIRECTORY” command, using the supplied file specification. Qualifiers may be included in the optional “par” tag to control the format of the listing.

```
<!--#dcl dir="web:[000000]" -->
<!--#dcl dir="web:[000000]" par="/nohead/notrail" -->
<!--#dcl dir="web:[000000]" par="/size/date" -->
```

- **EXEC** (privileged)

Execute the specified DCL command.

```
<!--#dcl exec="show device/full tape1:" -->
```

- **FILE** (privileged)

Execute the DCL command procedure specified as a VMS file path, with any specified parameters applied to the procedure.

```
<!--#dcl file="WASD_ROOT:[SHTML]TEST.COM" par="PARAM1 PARAM2" -->
```

- **VIRTUAL** (privileged)

Execute the DCL command procedure specified in URL-style syntax, with any specified parameters applied to the procedure.

```
<!--#dcl virtual="../../shtml/test.com" par="PARAM1 PARAM2" -->
```

- **CGI**

Execute the specified CGI script. The CGI response header is suppressed and only the response body is included in the document.

```
<!--#dcl cgi="/cgi-bin/calendar?2004" -->
```

4.5.5 #ECHO

The `#echo` directive incorporates the specified information into the processed document. Multiple tags may be used within the one directive.

- **VALUE=**
VAR=

Any SSI variable (e.g. `CREATED`), CGI variable (e.g. `HTTP_USER_AGENT`), or document assigned variable (e.g. `EXAMPLE1`), see Section 4.6.

```
<!--#echo value={created} var={example1} -->
```

- **CREATED**

The date/time of the current document’s creation.

```
<!--#echo created=["time-format"] -->
```

- **DATE_LOCAL**

Include the current date/time.

```
<!--#echo date_local["time-format"] -->
```

- **DATE_GMT**

Include the current Greenwich Mean Time (UTC) date/time.

```
<!--#echo date_gmt["time-format"] -->
```

- **DOCUMENT_NAME**

The current document's URL-style path.

```
<!--#echo document_name -->
```

- **FILE_NAME**

The current document's VMS file path.

```
<!--#echo file_name -->
```

- **HEADER**

Append the specified string to the response header (with correct carriage control). Should be used as early as possible in the SSI document.

```
<!--#echo header="Pragma: no-cache" -->
<!--#echo header="X-Extension-Header: just an example!" -->
```

- **LAST_MODIFIED**

The date/time of the current document's last modification.

```
<!--#echo last_modified["time-format"] -->
```

4.5.6 #ELIF

The *#elif* directive (else-if) allows blocks of HTML markup and SSI directives to be conditionally processed, see Section 4.7 and Section 4.5.14. This directive effectively allows a *case* statement to be constructed.

```
<!--#elif var="{variable}|literal" -->
```

4.5.7 #ELSE

The *else* directive allows blocks of HTML markup and SSI directives to be conditionally processed, see Section 4.7. It is the default block after an “*#if*”, “*#orif*” or “*#elif*”.

```
<!--#else -->
```

4.5.8 #ENDIF

The *#endif* directive marks the end of a block of document text being conditionally processed, see Section 4.7.

```
<!--#endif -->
```

4.5.9 #EXEC

The *#exec* directive executes a DCL command and incorporates the output into the processed document. It is the VMS equivalent of the *exec* shell directive of some Unix implementations. It is implemented in the same way as the *#DCL* directive, and so the general detail of that directive applies. It supports both the *cmd* tag and the *cgi* tag, allowing execution of CGI scripts (the response header is absorbed).

```
<!--#exec cmd="show device/full tape1:" -->
<!--#exec cgi="/cgi-bin/calendar?2004" -->
```

The *exec* directive is for *privileged* documents only, documents defined as those being owned by the SYSTEM account, and not being world-writeable. The reason for this should be obvious. There are implicit security concerns about any document being able to execute any DCL command(s), even if it is being executed in a completely unprivileged process.

4.5.10 #EXIT

The *#exit* directive causes the server to stop processing the current SSI file. If the current file was an *#included* SSI file, processing continues back with the parent file. Note that the *#stop* directive also is available, it stops processing of the entire virtual document.

```
<!--#exit -->
```

4.5.11 #FCREATED

The *#fcreated* directive incorporates the creation date/time of a specified file/document into the processed document.

- **FILE**

Document specified using a VMS file path.

```
<!--#fcreated file="file-name" [fmt="time-format"] -->
```

- **VIRTUAL**

Document specified using URL-style syntax.

```
<!--#fcreated virtual="path" [fmt="time-format"] -->
```

4.5.12 #FLASTMOD

The *#lastmod* directive incorporates the last modification date/time of a specified file/document into the processed document.

- **FILE**

Document specified using a VMS file path.

```
<!--#lastmod file="file-name" [fmt="time-format"] -->
```

- **VIRTUAL**

Document specified using URL-style syntax.

```
<!--#lastmod virtual="path" [fmt="time-format"] -->
```


4.5.13 #FSIZE

The *#size* directive incorporates the size, in bytes, kbytes or Mbytes, of a specified file/document into the processed document.

- **FILE**

Document specified using a VMS file path.

```
<!--#size file="file-name" [fmt="size-format"] -->
```

- **VIRTUAL**

Document specified using URL-style syntax.

```
<!--#size virtual="path" [fmt="size-format"] -->
```

4.5.14 #IF

The *#if* directive allows blocks of HTML markup and SSI directives to be conditionally processed, see Section 4.7.

- **VAR=**

Variable the decision will be based upon.

```
<!--#if var="{variable}|literal]" -->
```

- **EQS=**

Is the string the same as in the variable?

- **EQ=**

If the variable is a number is it the same as this?

- **GT=**

If the variable is a number is it greater than this?

- **LT=**

If the variable is a number is it less than this?

- **SRCH=**

Search the variable for this string. May contain the “*” wildcard, matching one or more characters, and the “%”, matching any single character.

As in the following examples:

```
<!--#if value={DOCUMENT_URI} eqs="/wasd_root/doc/env/xssi.shtml" -->
<!--#if value={COUNT} lt=10 -->
<!--#if value="This is a test!" eqs={STRING} -->
<!--#if value={PATH_INFO} srch="*/env/*" -->
```

4.5.15 #INCLUDE

The *#include* directive incorporates the contents of a specified file/document into the processed document.

- **FILE**

Include the contents of the document specified using a VMS file specification.

```
<!--#include file="file-name" -->
```

- **VIRTUAL**

Include the contents of the document specified using URL-style syntax.

```
<!--#include virtual="path" -->
```

The contents of the specified file are included differently depending on the MIME content-type of the file. Files of *text/html* content-type (HTML documents) are included directly, and any HTML tags within them contribute to the markup of the document. Files of *text/plain* content-type (plain-text documents) are encapsulated in “<PRE></PRE>” tags and have all HTML-forbidden characters (e.g. “<”, “&”) escaped before inclusion in the processed document. An HTML file can be forced to be included as plain-text by using the following syntax:

```
<!--#include virtual="example.html" type="text/plain" -->
```

To “force” a file to be considered as text regardless of the actual content (as determined by the server from the file type), use one of the following depending on whether it should be rendered as plain or HTML text.

```
<!--#include virtual="example.html" content="text/plain" -->
<!--#include virtual="example.html" content="text/html" -->
```

Other SSI files may be included and their content dynamically included in the resulting document. To prevent a recursive inclusion of documents the nesting level of SSI documents is limited to five.

4.5.16 #MODIFIED

The *#modified* directive allows a document author to control the “Last-Modified:”/“If-Modified-Since:”/“304 Not modified” behaviour of an SSI document. See Section 4.1.

- **no tag**

Get the last-modified date/time of the current document.

```
<!--#modified -->
```

- **FILE**

Get the last-modified date/time of the document specified using VMS file specification.

```
<!--#modified file="file-name" -->
```

- **VIRTUAL**

Get the last-modified date/time of the document specified using URL-style syntax.

```
<!--#modified virtual="path" -->
```

- **IF-MODIFIED-SINCE**

Compares any “If-Modified-Since:” request header timestamp to the revision date time obtained using *file* or *virtual* (most recent if multiple). If the document timestamp is more recent (has been modified) an appropriate “Last-Modified” response header field is generated and added to the response, and document processing continues. If it has not been modified a “304” response header is return (document not modified) and document processing stops.

```
<!--#modified if-modified-since -->
```

- **LAST-MODIFIED**

Adds a “Last-Modified:” response header field using a timestamp retrieved using *file* or *virtual* (note: unnecessary if the *if-modified-since* tag is used).

```
<!--#modified last-modified -->
```

- **EXPIRES**

Adds a “Expires:” response header field. The string literal should be a legitimate RFC-1123 date string. This can be used for pre-expiring documents (so they are always reloaded), set it to a date in the not-too-distant past (as in the example below). Of course it could also be used for setting the legitimate future expiry of documents.

```
<!--#modified expires="Fri, 13 Jan 1978 14:00:00 GMT" -->
```

4.5.17 #ORIF

The *#orif* directive (or-if) allows blocks of HTML markup and SSI directives to be conditionally processed, see Section 4.7 and Section 4.5.14. In the absence of any real expression parser this directive allows a block to be processed if one of multiple conditions are met.

```
<!--#orif var="{variable}|literal" -->
```

4.5.18 #PRINTENV

The *#printenv* directive prints a plain-text list of all SSI-specific, then CGI, then document-assigned variables (see Section 4.6). This directive is intended for use when debugging flow-controlled SSI documents.

```
<!--#printenv -->
```

The following link uses the example SSI document, WASD_ROOT:[DOC.ENV]XSSI.SHTML, to demonstrate this.

online demonstration

4.5.19 #SET

The *#set* directive allows a user variable to be assigned or modified, see Section 4.6.

```
<!--#set var="variable-name" value="whatever" -->
```

Variables are always stored as strings and have a finite but generally usable length. Some comparison tags provided in the flow-control directives treat the contents of variables as numbers. A numeric conversion is done at evaluation time.

4.5.20 #SSI

The `#ssi` directive allows multiple SSI directives to be used without the requirement to enclose them in the normal HTML comment tags (i.e. `<!-- -->`). This helps reduce the clutter in an SSI document that uses the extended capabilities of variable assignment and flow control. Document HTML cannot be included between the opening and closing comment elements of the `"#ssi"` tag, although of course document output can be generated using the `"#echo"` tag.

```
<!--#ssi
#set var=HOUR value={DATE_LOCAL,12,2}
#if var={HOUR} lt=12
  #set var=GREETING value="Good morning"
#elif var={HOUR} lt=19
  #set var=GREETING value="Good afternoon"
#else
  #set var=GREETING value="Good evening"
#endif
-->
```

The example SSI document, `WASD_ROOT:[DOC.ENV]XSSI.SHTML`, illustrates this concept.

4.5.21 #STOP

The `#stop` directive causes the server to stop processing the virtual document. It can be used with flow control structures to conditionally process only part of a virtual document. Note that the `#exit` directive also is available, it stops processing of the current file (for nested `#includes`, etc.).

```
<!--#stop -->
```

4.6 Variables

The SSI processor maintains information about the server, date and time, request path, request parameters, etc., accessible via *variable name*. Although these *server variables* cannot be modified by the document the processor also allows the author to create and assign new *document variables* by name. SSI variables have global scope, with a small number of exceptions listed below. That is, the same set of variables are shared with the parent document by any other SSI documents *#included*, and any included by those, etc.

Local variables:

- `DOCUMENT_DEPTH`, the current nesting level for `#included` SSI files
- `PARENT_FILE_NAME`, if an `#included` SSI file the name of the including file
- `THIS_FILE_NAME`, the name of the SSI file currently being processed

One other special-purpose variable, `THE_FILE_NAME`, see Section 4.9.1.

Server assigned variables comprise some SSI-specific as well as the same CGI variables available to CGI scripts.

These may be found listed in the *CGI Scripting* chapter of the *Technical Overview*.

[online demonstration](#)

Whenever a directive uses information from a tag (see Section 4.4) values from variables may be substituted as a whole or partial value. This is done using curly braces to delimit the variable name. For example

```
<!--#include virtual={FILENAME} -->
```

would include the file named by the contents of a variable named “FILENAME”. When using a variable in a tag it is not necessary to enclose the tag parameter in quotation marks unless there is additional literal text. Variables may also be used within literal strings, producing a compound, resultant string, as in the following example

```
<!--#echo var="Hello {REMOTE_HOST}, time here is {LOCAL_TIME}" -->
```

Variables are considered numeric when they begin with a digit. Those beginning with an alphabetic are considered to have a numeric value of zero.

Variables are considered to be boolean *false* if empty and *true* when not empty.

Substrings

It is also possible to extract substrings from variables using the following syntax,

```
{variable-name,start-index,count}
```

where the start-index begins with the zeroth character and numbers up to the last character in the string, and count may be zero or any positive number. If only one number is supplied it is regarded as a count and the string is extracted from the zeroth character.

To illustrate,

```
<!--#set var=EXAMPLE value="This is an example!" -->
<!--#echo "{EXAMPLE,2}at was {EXAMPLE,8,999}" -->
```

would output

```
That was an example!
```

Other “Functions”

- **LENGTH** - This “function” returns the length of the parameter string (or substring).

```
{variable-name[,start-index],count]],length}
```

For example

```
<!--#set var=EXAMPLE value="This is an example!" -->
<!--#echo "\"{EXAMPLE}\" is {EXAMPLE,length} characters long." -->
<!--#echo "\"{EXAMPLE,5,2}\" is {EXAMPLE,5,2,length} characters long!" -->
```

would output

```
"This is an example!" is 19 characters long.
"is" is 2 characters long!
```

- **EXISTS** - This “function” returns *true* if the variable exists and *false* if it does not. This is useful as accessing a non-existent variable will result in an SSI error message with document processing ceasing!

```
var={variable-name,exists}
```

For example

```
<!--#set var=BOGUS_VARIABLE value="irrelevant" -->
<!--#if var={BOGUS_VARIABLE,exists} -->
    &quot;BOGUS_VARIABLE&quot; exists!
<!--#else -->
    &quot;BOGUS_VARIABLE&quot; does NOT exist!
<!--#endif -->
```

Example

The example SSI document, `WASD_ROOT:[DOC.ENV]XSSI.SHTML`, illustrates these concepts.

4.7 Flow Control

WASD SSI allows blocks of document to be conditionally processed. This uses constructs in a similar way to any programming language. The emphasis has been on simplicity and speed of processing. No complex expression parser is provided. Despite this, complex document constructs can be implemented. Flow control structures may be nested up to eight levels.

- **#if** - Marks the start of a conditionally processed block. If evaluated true the block is processed.
- **#orif** - Allows a document block to be processed if one of multiple conditions are met. Must be used immediately following a “#if” or “elif”. (This is the only really WASD-idiosyncratic element)
- **#elif** - Allows a series of conditionals to be tested each with its own document block available for processing. Allows a type of case statement to be constructed.
- **#else** - Provides a default document block following unsuccessful “#if”, “orif” and “elif” testing and consequent non-processed blocks.
- **#endif** - Terminates a conditional block.

The “#if”, “#orif” and “#elif” directives must provide an evaluation. This can be single variable, which if numeric and non-zero is considered true, if zero if false, or can be a string, which if empty is false, and if not empty is true. Tests can be made against the variable which when evaluated return a true or false. Multiple tests may be made against the one variable, or against more than one variable. Multiple tests act as a logical AND of the results and terminate when the first fails.

- **eqs** - If the supplied string is the same as the variable string.
- **srch** - If the variable string matches the supplied search string. The search string may contain the “*”, matching any zero or more characters, and “%”, matching any one character.
- **eq** - If the numeric value of the variable is the same as that of the supplied number. For a numeric value test to be legitimate both values must begin with a digit. Those beginning with an alphabetic are considered zero.

- **lt** - If the numeric value of the variable is less than that of the supplied number.
- **gt** - If the numeric value of the variable is greater than that of the supplied number.

Any evaluation can have the result negated by prefixing it with an exclamation point. For instance, the first of these examples would produce a *false* result, the second *true*.

```
<!--#if value="test" !eqs="test" -->
<!--#if value=20 !lt=10 -->
```

The following is a simple example illustration of variable setting, use of variable substrings, and conditional processing of document blocks.

```
<!--##config trace=1 -->
<HTML>
<!--#set var=HOUR value={DATE_LOCAL,12,5} -->
<!--#if var={HOUR} lt=12 -->
<!--#set var=GREETING value="Good morning" -->
<!--#elif var={HOUR} lt=19 -->
<!--#set var=GREETING value="Good afternoon" -->
<!--#else -->
<!--#set var=GREETING value="Good evening" -->
<!--#endif -->
<HEAD>
<TITLE><!--#echo var={GREETING} -->
<!--#echo var="{REMOTE_HOST}!" --></TITLE>
</HEAD>
<BODY>
<H1>Simple XSSi Demonstration</H1>
<!--#echo var={GREETING} --> <!--#echo var={REMOTE_HOST} -->,
the time here is <!--#echo var={DATE_LOCAL,12,5} -->.
<!--#if var={REMOTE_HOST} eqs={REMOTE_ADDR} -->
(Sorry, I do not know your name, DNS lookup must be disabled!)
<!--#endif -->
</BODY>
</HTML>
```

The example SSI document, `WASD_ROOT:[DOC.ENV]XSSI.SHTML`, further illustrates these concepts.

4.8 Query Strings

A query string may be passed to an SSI document in much the same way as to a CGI script. In this way the behaviour of the document can be varied in accordance to information explicitly passed to it when accessed. To prevent the server's default query engine being given the request precede any query string with “`?httpd=ssi`”. The server detects this and passes the request instead to the SSI processor. Just append the desired query string components to this as if they were form elements. For example:

```
?httpd=ssi&printenv=no
?httpd=ssi&printenv=yes
?httpd=ssi&trace=yes&test2=one&test2=two&test3=three
```

The following link uses the example SSI document, `WASD_ROOT:[DOC.ENV]XSSI.SHTML`, to demonstrate this. Look for the “`FORM_TEST1=one`”, etc.

[online demonstration](#)

4.9 File and Virtual Specifications

Documents may be specified using either the “FILE” or “VIRTUAL” tags.

The “FILE” tag expects an absolute VMS file specification.

The “VIRTUAL” tag expects an URL-style path to a document. This can be an absolute or relative path. See Section 2.3 for further details.

4.9.1 THE_FILE_NAME

Generally, when an error are encountered document processing halts and and an error report is generated. For some common circumstances, in particular the existence or not of a particular file, may require an alternative action. For file activities (e.g. `#include`, `#lastmod`, `#created`, `#fsize`) the optional `fmt=""` tag provides some measure of control on error behaviour. If the format string begins with a “?” files not found are not reported as errors and processing continues. Other file systems errors, such as directory not found, syntax errors, etc., are always reported.

Every time a file is accessed (e.g. `#include`, `#lastmod`) the server variable `THE_FILE_NAME` gets set to that name if successful, or reset to empty if unsuccessful. This variable can be checked to determine success or otherwise.

- For `#included` files, the `'fmt="?"` just suppresses an error report, if the file exists then it is included.
- For `#modified` file specifications use `'fmt="?"` to suppress error reporting on evaluation of files that may exist but are not mandatory.
- For file statistic directives (e.g. `#lastmod`, `#fcreated`, `#fsize`) the `'fmt="?"` tag completely suppresses all output as well as error reporting. This can be used to check for the existence of a file. For example if the file `TEST.TXT` exists in the following example the variable `THE_FILE_NAME` would contain the full file name, if it does not exist it would be empty, and the code example would behave accordingly.

```
<!--#fcreated virtual="TEST.TXT" fmt="?" -->
<!--#if var={THE_FILE_NAME} eqs="" -->
File does not exist!
<!--#else -->
File exists!
<!--#endif -->
```

4.10 Time Format

Whenever a time directive is used an optional tag can be included to specify the format of the output. The default looks a little VMS-ish. If a format specification is made it must conform to the C programming language function `strftime()`.

The format specifier follows a similar syntax to the C standard library `printf()` family of functions, where conversion specifiers are introduced by percentage symbols. Here are some example uses:

```
The date is <!--#echo date_local fmt="%d/%m/%y" -->.
The time is <!--#echo date_local fmt="%r" -->.
The day-of-the-week is <!--#echo date_local fmt="%A" -->.
```


A problem with any supplied time formatting specification will be reported.

The following table provides the general conversion specifiers. For further information on the formatting process refer to a C programming library document on the *strftime()* function.

strftime() Format Directives

Specifier	Replaced by
a	The locale's abbreviated weekday name
A	The locale's full weekday name
b	The locale's abbreviated month name
B	The locale's full month name
c	The locale's appropriate date and time representation
C	The century number (the year divided by 100 and truncated to an integer) as a decimal number (00 - 99)
d	The day of the month as a decimal number (01 - 31)
D	Same as %m/%d/%y
e	The day of the month as a decimal number (1 - 31) in a 2 digit field with the leading space character fill
Ec	The locale's alternative date and time representation
EC	The name of the base year (period) in the locale's alternative representation
Ex	The locale's alternative date representation
EX	The locale's alternative time representation
Ey	The offset from the base year (%EC) in the locale's alternative representation
EY	The locale's full alternative year representation
h	Same as %b
H	The hour (24-hour clock) as a decimal number (00 - 23)
I	The hour (12-hour clock) as a decimal number (01 - 12)
j	The day of the year as a decimal number (001 - 366)
m	The month as a decimal number (01 - 12)
M	The minute as a decimal number (00 - 59)
n	The newline character
Od	The day of the month using the locale's alternative numeric symbols
Oe	The date of the month using the locale's alternative numeric symbols

Specifier	Replaced by
OH	The hour (24-hour clock) using the locale's alternative numeric symbols
OI	The hour (12-hour clock) using the locale's alternative numeric symbols
Om	The month using the locale's alternative numeric symbols
OM	The minutes using the locale's alternative numeric symbols
OS	The seconds using the locale's alternative numeric symbols
Ou	The weekday as a number in the locale's alternative representation (Monday=1)
OU	The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols
OV	The week number of the year (Monday as the first day of the week) as a decimal number (01 -53) using the locale's alternative numeric symbols. If the week containing January 1 has four or more days in the new year, it is considered as week 1. Otherwise, it is considered as week 53 of the previous year, and the next week is week 1.
Ow	The weekday as a number (Sunday=0) using the locale's alternative numeric symbols
OW	The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols
Oy	The year without the century using the locale's alternative numeric symbols
p	The locale's equivalent of the AM/PM designations associated with a 12-hour clock
r	The time in AM/PM notation
R	The time in 24-hour notation (%H:%M)
S	The second as a decimal number (00 - 61)
t	The tab character
T	The time (%H:%M:%S)
u	The weekday as a decimal number between 1 and 7 (Monday=1)
U	The week number of the year (the first Sunday as the first day of week 1) as a decimal number (00 - 53)
V	The week number of the year (Monday as the first day of the week) as a decimal number (00 - 53). If the week containing January 1 has four or more days in the new year, it is considered as week 1. Otherwise, it is considered as week 53 of the previous year, and the next week is week 1.
w	The weekday as a decimal number (0 [Sunday] - 6)
W	The week number of the year (the first Monday as the first day of week 1) as a decimal number (00 - 53)
x	The locale's appropriate date representation

Specifier	Replaced by
X	The locale's appropriate time representation
y	The year without century as a decimal number (00 - 99)
Y	The year with century as a decimal number
Z	Timezone name or abbreviation. If timezone information is not available, no character is output.
%	%

4.11 OSU Compatibility

Essential compatibility with OSU Server Side Includes directives is provided. This is intended to ease any transition to WASD, as existing SSI documents will not need to be changed unless any of the WASD capabilities are required. To provide transparent processing of OSU “.HTMLX” files ensure the following WASD configuration is in place.

In HTTPD\$CONFIG file:

```
[AddType]
.HTMLX  text/x-shtml  -  OSU SSI HTML
```

Note that the content description must contain the string "OSU" to activate some compliancy behaviours.

In HTTPD\$MAP file:

```
redirect /*.*.htmlx /*.htmlx?httpd=ssi&__part=*
```

This provides a mechanism for the OSU part-document facility. (Yes, the “__part” has two leading underscores!)

OSU Directives

The following OSU directives are provided specifically for OSU compatibility, although there is no reason why most of these may not also be deployed in general WASD SSI documents if there is a requirement. Note that these are OSU-specifics, other OSU directives are provided by the standard WASD SSI engine.

OSU Compatible Directives

Directive	Description
#begin <i>label</i> [<i>label</i>]	delimit a part-document (see OSU Parts)
#config verify=1	enable commented-tag trace output
#echo accesses	document access count

Directive	Description
#echo accesses_ordinal	document access count
#echo getenv=""	output logical or symbol
#echo hw_name	system hardware name
#echo server_name	HTTPd server host name
#echo server_version	HTTPd software version
#echo vms_version	HTTPd system version of VMS
#end label [label]	delimit a part-document (see OSU Parts)
#include [file virtual]="" part="label"	include only part of a virtual document

If WASD is configured for OSU SSI compatibility the following link provides an online demonstration as well as further explanation of the OSU SSI engine using an OSU preprocessor document from the distribution (included within copyright compliance).

[online demonstration](#)

How do we know WASD is processing it? Look for the *#echo var="GETENV=SYS\$REMOTE_ID"* towards the end of the document. It should indicate "[VARIABLE_DOES_NOT_EXIST!]" because it's attempting to output a DECnet-related logical name!

OSU "Part"s

The OSU processor allows for delimited subsections of an *#included* document, or a URL referenced document for that matter, to be included in the output. This is supported, but only for compatibility. It is only enabled for ".HTMLX" documents and if otherwise used may interact unexpectedly with WASD SSI flow-control.

4.12 Script-Generated SSI Documents

It is possible to have script output passed back through the SSI engine for markup. This approach might allow script output to automatically be wrapped in standard site headers and footers for example. Essentially the script must output an SSI-markup response body and include in the otherwise standard CGI response header a field containing "Script-Control: X-content-handler=SSI". The following example in DCL show the essential elements of such a script.

```
$ say = "write sys$output"
$ say "Status: 200"
$ say "Script-Control: X-content-handler=SSI"
$ say ""
$ say "<HTML>"
$ say "<HEAD>"
$ say "<TITLE>Example of X-content-handler=SSI</TITLE>"
$ say "</HEAD>"
$ say "<BODY>"
$ say "<!--#include virtual=\"/site/header.html\" -->"
$ say "<H1>Example of X-content-handler:SSI</H1>"
$ say "Hi there <!--#echo var=\"WWW_REMOTE_HOST\" -->"
$ say "<!--#include virtual=\"/site/footer.html\" -->"
$ say "</BODY>"
$ say "</HTML>"
```

Chapter 5

Clickable Image Support

Clickable image support is provided as an integral part of HTTPd functionality (i.e. it does not require script execution), and so can be quite efficient. It will process both NCSA and CERN configuration formats (in the same file if necessary, although for clarity that should be avoided).

Digression . . . How It Works

When the image specified in the anchor is clicked upon the browser sends a mapping configuration file URL, specified in the HTML anchor, along with the pixel coordinate of the click, as a query string, to the HTTPd server. The server interprets region specifications in the configuration file to determine which region corresponds to the coordinates in the query string. A matching specification's URL, or a default if none match, is then accessed by the server (if local), or sent back to, and then transparently reaccessed by the browser (redirected, if a different protocol or host).

Steps For Using a Clickable Image

1. create an *image configuration file* (see Section 5.1), mapping pixel coordinates of regions within the image to URLs
2. specify an HTML anchor using an inline "<IMG...>" tag, the "HREF=" specifies the path to the image configuration file
3. specify "ISMAP" in the "<IMG...>" tag

For example:

```
<A HREF="ismap_demo.ismap">  
<IMG SRC="ismap_demo.gif" ISMAP></A>
```

5.1 Image Configuration File

Image configuration is done using a plain-text file containing region keywords specifying image pixel coordinates and associated URLs. Clicking within these coordinates results in the corresponding URL being returned. Four keywords defining geometrically shaped series of coordinates are provided, along with a default keyword. These can be supplied in either of two formats. The NCSA format may be more commonly used.

1. NCSA

- **circle URL $x1,y1$ $x2,y2$**
URL to be returned when the click is within a circle of centre-point $x1,y1$ and a circumference specified by the edge-point $x2,y2$.
- **rectangle URL $x1,y1$ $x2,y2$**
URL to be returned when the click is within a rectangle having opposite corners $x1,y1$ and $x2,y2$.
- **point URL x,y**
With multiple points specified, the URL of the point closest to the click.
- **polygon URL $x1,y1$ $x2,y2$ $x3,y3$ [... xn,yn]**
URL to be returned when the click is within an arbitrary polygon having adjacent vertices specified by the series of coordinates $(x1,y1)$ through to (xn,yn) . If the polygon is not explicitly closed it is treated as if the first and last coordinates were connected.
- **default URL**
URL to be returned when the click is not within any of the specified coordinates, and there has been no *point* specified.

2. CERN

- **circle (x,y) r URL**
URL to be returned when the click is within a circle of radius r at centre-point (x,y) .
- **rectangle ($x1,y1$)($x2,y2$) URL**
URL to be returned when the click is within a rectangle having opposite corners $(x1,y1)$ and $(x2,y2)$.
- **point ($x1,y1$) URL**
With multiple points specified, the URL of the point closest to the click (strictly speaking, this is NCSA only).
- **polygon ($x1,y1$)($x2,y2$)...(xn,yn) URL**
URL to be returned when the click is within an arbitrary polygon having adjacent vertices specified by the series of coordinates $(x1,y1)$ through to (xn,yn) . If the polygon is not explicitly closed it is treated as if the first and last coordinates were connected.
- **default URL**
URL to be returned when the click is not within any of the specified coordinates, and there has been no *point* specified.

For online examples of rule usage within configuration files see Section 5.2 below. Note that:

- There must be only one region keyword on each line.

- The region keywords are scanned from first towards last, the first one with coordinates encompassing the click having the URL returned. Region coordinates within other regions should be defined first.
- Either full URLs (*protocol://host/path*) or partial URLs (*/path*) may be specified against the keywords. As an extension to NCSA and CERN capability, this image mapper will also accept relative URLs (*path*, *../path*, etc.).
- The image mapping utility may return textual error messages if the configuration keywords or parameters are incorrect.
- The keywords may be abbreviated to *circ*, *rect*, *poly*, *poin* and *def*.
- Blank lines are ignored.
- Commentary must be preceded by a “#” or “!” character.

hint . . .

To establish the region keywords and coordinates required for the configuration file it may be necessary to use a program such as “XV” to display the image, then by using the mouse locate the required parts of the image, reading off and noting the coordinate pairs, and finally using these to compose the configuration file.

5.2 Examples

See example in the WASD_ROOT:[EXERCISE] directory.

Chapter 6

Document Searching

The *query* and *extract* scripts provide real-time searching of plain-text and HTML documents, and document retrieval. The search is a simple-string search, not a GREP-style search. It is designed to provide a useful mechanism for locating documents containing a keyword, not for document analysis. It has the useful feature for plain-text documents of allowing the selective extraction of only the portion near the *hit*.

Only files with a plain-text or HTML MIME data type (see Chapter 2) will be searched. Others may be specified, or be selected from wildcard file specification, but they will not actually have their contents searched.

Directory specifications may include a wildcard elipsis (allowing a directory tree to be traversed) and/or file name wildcards. In other words, anything acceptable as VMS file system syntax (except in URL-format of course). See examples in Section 6.3.2.

6.1 Plain-Text Search

A search of a plain-text file is straight-forward. Each line in the file is searched for the required string. The first time it is encountered is considered a *hit*. The line is not searched for any further occurrences.

Searches of plain text files allow the subsequent selection of partial documents (i.e. the retrieval of only a number of lines around any actual hit). This allows the user to selectively extract a portion of a document, avoiding the need to explicitly scan through to the section of interest.

6.2 HTML Search

A search of an HTML file is a little more complex. As might be expected, only text presented in the document text is searched, markup text is ignored. That is, all text not part of an HTML *tag* construct is extracted and searched. For example, out of the following HTML fragment

```
<!-- an example HTML document -->
<P>
The document entitled <A HREF="example.html">"Example Document"</A>
provides only an <I>overview</I> of the full capabilities of HTML.
```

only the following text would actually be searched

```
The document entitled "Example Document" provides only an overview
of the full capabilities of HTML.
```

The mechanism for partial document retrieval available with plain-text files is **not** present with HTML documents. HTML files generally must be treated as a whole, with the formatting of current sections often very dependent on the formatting of previous sections. This makes extracting a subsection perilous without extensive syntactical analysis. On the positive side, HTML documents tend to be already divided into meaningful subdocuments (files), making retrieval of a hit naturally more-or-less within context.

Instead of partial document retrieval, the document is processed to place anchors for each hit, making it possible to jump directly to a particular section of interest. Generally this works well but may occasionally distort the presentation of a document.

6.3 Search Syntax

A search may be initiated in basic three ways:

1. Appending a question-mark and search string to a file specification (the simple syntax of “ISINDEX”-style searching). This is standard HTTP, and of course must conform to HTTP syntax.
2. Providing the name of the query script followed by the directory path to be searched. The script then returns a standard search form.
3. *Forms*-based search, which allows the format and mechanism of the search to be controlled.

6.3.1 “ISINDEX” Search

Placing the HTML tag “<ISINDEX>” within a document’s text is sufficient to inform the browser that searching is available for that document. The browser will inform the user of this and allow a search of that document to be initiated at any time. Note that it is limited to the one document.

Using the keyword search syntax explicitly is another method of initiating a search, and additionally can use a wildcard in the document specification. For example:

```
/wasd_root/doc/env/*.??formatted
```

[online demonstration](#)

6.3.2 Standard Search Form

Using the “QUERY” script name followed by a URL-format path specifying the directory to be searched returns a standard, script-generated search form.

[online demonstration](#)

As with all search specifications, the directory specification may include wildcard a elipsis (allowing a directory tree to be traversed) and/or file name wildcards. In other words, anything acceptable as VMS file system syntax (except in URL-format of course). See the following examples.

[online demonstration](#)

6.3.3 Forms-Based Search

A “forms-based” search is initiated by the server receiving a file specification, which of course may contain wildcards, followed by a *search* parameter. This is a typical HTML *forms* format URL. For example:

```
*.txt?search=SIMPLE
/web/.../*.*?search=THIS
sub_directory/*.*?search=THAT
../sibling_directory/*.HTML?search=OTHER
```

[online demonstration](#)

6.3.4 Search Options

Additional URI components may be appended after the initial “search=” parameter. These are appended with intervening “&”) characters.

- **Case-Sensitivity.** An optional URI component of “case=yes” or “case=no” makes the search case-sensitive or case-insensitive (the default). The following example illustrates the use of this syntax:

```
/web/html/.../*.html?search=Protocol&case=yes
/web/html/.../*.html?search=PrOtOcOl&case=no
```

[online demonstration](#)

- **Hits.** An optional URI component of “hits=document” or “hits=line” makes the search results be presented by-document (file) or by line-by-line (the default). The following example illustrates the use of this syntax:

```
/web/html/.../*.html?search=protocol&hits=document
/web/html/.../*.html?search=protocol&hits=line
```

[online demonstration](#)

6.3.5 Example Search Form

To allow the client to enter a search string and submit a search to the server a HTML level 2 *form* construct can be used. Here is an example:

```
<FORM ACTION="/web/html/.../*.html">
Search HTML documents for:
<INPUT TYPE=text NAME="search">
<INPUT TYPE=submit VALUE="[execute]">
</FORM>
```

[online demonstration](#)

Bells and Whistles

A form providing all the options referred to in Section 6.3.4 is shown below (some additional white-space introduced for clarity):

```
<FORM ACTION="/web/html/.../*.html">

Search HTML documents for:
<INPUT TYPE=text NAME="search">
<INPUT TYPE=submit VALUE="[execute]">

<BR><TT><A HREF="/query/-/aboutquery.html">About</A> this search.</TT>

<BR><TT>Output By:
line <INPUT TYPE=radio NAME="hits" VALUE="line" CHECKED>
document <INPUT TYPE=radio NAME="hits" VALUE="document"></TT>

<BR><TT>Case Sensitive:
no <INPUT TYPE=radio NAME="case" VALUE="no" CHECKED>
yes <INPUT TYPE=radio NAME="case" VALUE="yes"></TT>

</FORM>
```

[online demonstration](#)

Chapter 7

VMS Help and Text Libraries

Affectionately known as *Conan the Librarian*, and with all due acknowledgement to Wierd Al.Yankovic “:^”), this script makes VMS *Help* and *Text* libraries accessible in the hypertext environment.

The librarian script will be automatically activated if the file specified has an extension of “.HLB” or “.TLB”. Alternatively it may be explicitly activated by specifying */conan* as a prefix to the file specification (but the ability to provide a relative specification is lost). The following examples illustrate the syntax:

```
<A HREF="/sys$common/syshlp/helpplib.hlb">VMS help</A>  
<A HREF="/conan/sys$common/syshlp/helpplib.hlb">VMS help via /Conan</A>
```

[online demonstration](#)

Other Librarian Functionality

To obtain an index of matching libraries explicitly activate the *Conan* script providing a wildcard file specification.

```
<A HREF="/conan/sys$common/syshlp/*.hlb">All Help libraries in SYS$HELP</A>
```

[online demonstration](#)

To obtain the library header information add a query string to the library file specification, as shown in the following example:

```
<A HREF="/conan/sys$common/syshlp/helpplib.hlb?do=header">VMS help library header</A>
```

[online demonstration](#)

Chapter 8

Bookreader Books and Libraries

Access to DEC's Bookreader format documentation (and any generated internally) is provided in the WASD hypertext environment via two integrated scripts, *HyperReader*, which reads the books, and *HyperShelf*, which reads the Library and Shelf structures.

The HyperReader and HyperShelf scripts are automatically activated when the document file's extension is ".DECW\$BOOK" and ".DECW\$BOOKSHELF" respectively. Alternatively, the respective scripts may be explicitly specified (but the ability to provide a relative specification is lost).

If the server system supports Bookreader documentation collection the following link will provide an online demonstration:

[online demonstration](#)

Chapter 9

Web Document Update

The **Update** facility allows Web documents and file environments to be administered from a standard browser. This facility is available to Web administrator and user alike. Availability and capability depends on the authorization environment within the server.

It **should be stressed** that this is not designed as a full hypertext administration or authoring tool, and for document preparation relies on the editing capabilities of the “<TEXTAREA>” widget of the user’s browser. It does however, allow **ad-hoc changes** to be made to documents fairly easily, as well as allowing documents to be deleted, and directories to be created and deleted.

Consult the current **Update** documentation for usage detail.

[online hypertext link](#)

Update Access Permission

Of course, the user must have write (POST/PUT) access to the document or area on the server (i.e. the *path*) and the server account have file system permission to write into the parent directory.

Contact the Web Administrator for further information on the availability of authentication and authorization permissions to do online updates of Web paths.

The server will report “Insufficient privilege or object protection violation ... /path/” if it does not have file system permission to write into a directory.

Write access by the server into VMS directories (using the POST or PUT HTTP methods) is controlled using VMS ACLs. **This is in addition to the path authorization of the server itself of course!** The requirement to have an ACL on the directory prevents inadvertant mapping/authorization of a path resulting in the ability to write somewhere not intended.

Two different ACLs implement two grades of access.

1. If the ACL grants **CONTROL** access to the server account then only VMS-authenticated usernames with security profiles can potentially write to the directory. Only potentially,

because a further check is made to assess whether that VMS account in particular has write access.

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

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

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

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

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

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

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

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

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