Xinet API Guide Version 17.7

Xinet Version 17.7
Release Date: August 2014

Introduction

This guide provides information about how to communicate with Xinet using the *portalDI* CGI. The *portalDI* CGI is a query-based application that takes over for and streamlines the functionality of a set Xinet CGIs upon which Xinet Portal formerly depended for such things as:

- Constraining information displays
- Displaying *Browse* and *Search* results
- Displaying and manipulating *Shopping Baskets*
- Displaying and interacting with metadata
- Archiving and Restoring files
- Customizing Image Order
- Customizing Uploads
- Customizing Solr Searching

Xinet provides information in this *Guide* about the *portalDI* binary so that developers can customize and extend the Xinet beyond those options which ship with the software. This CGI is platform- and program-independent and also faster than the CGIs Xinet Portal previously called upon.

1.1 Who should read this document?

The *Xinet API Guide* has been written for programmers who want to integrate third party software with a Xinet workflow and for those who want to create custom widgets that extend Xinet functionality.

1.2 Conventions

We use several typographical conventions in this manual to help readers distinguish what must be typed as is and what is simply a parameter to be substituted. In particular, readers should type text set in a typewriter-like font, `like this`, *literally*. For example:

```
# more README or c:\> type README
```

Readers should enter this command exactly as shown above (minus the shell prompt on UNIX systems, #, or the `c:\>` prompt on Windows systems). Readers should substitute their own applicable text for text set in slanted typewriter type, *`like this`*. For example, when one sees

`% ls` *`directory_name`* or `C:\> dir` *`directory_name`*

one might type:

`% ls doc` or `C:\> dir doc`

1.2.1 On UNIX systems

The two examples given above illustrate another convention: when a command must be issued by the superuser (*root*), you will see the prompt `#`; when a command may be issued by a normal user, you will see the prompt `%`. As in most documentation for UNIX software, text references to UNIX functions have the form *more*(1). The word in italic type is the name of the function; the parenthesized number is the section of the *UNIX Programmer's Manual* containing the function's manual page.

1.2.2 On Windows systems

When the manual discusses locations for files and programs, it uses default installation path names. If you install files and programs in other locations, the paths will not be the same as in this manual.

Where to look for technical information about Xinet products

Xinet technical information spans several volumes, with each software distribution having an appropriate guide. The guides are available for downloading in the *Maintenance* section of the Xinet Web site, *www.xinet.com.* The Web site also provides links for ordering printed copies.

To help you easily find what you need, the guides are extensively indexed and cross-referenced, and whenever possible, the index entries and cross references provide live links to the appropriate pages within the PDF.

Xinet technical documentation includes:

- **Xinet API Guide** (this guide)

  This guide provides information about how to communicate with Xinet using the portalDI CGI. The portalDI CGI is a query-based application that takes over for and streamlines the functionality of a set Xinet CGIs upon which Xinet Portal formerly depended for such things as:

  - Constraining information displays
  - Displaying Browse and Search results
  - Displaying and manipulating Shopping Baskets
  - Displaying and interacting with metadata
  - Archiving and Restoring files
  - Customizing Image Order
  - Customizing Uploads
  - Customizing Solr Searching

  Xinet provides information in this guide about the portalDI binary to help developers customize and extend the Xinet beyond those options which ship with the software.

- The *Xinet Administration Guide*

  This guide provides basic information about installation, licensing and Xinet administration. It also provides information to consider before installation, such as optimal file-system organization and security issues. Chapters include details about administrative settings and basic ways to customize the look and feel of Web sites.

  The guide includes information about setting up everyday management of the Xinet database, including ways to customize data fields and control their use by users and groups. It also contains information about automating production activities based on changes in the database and information about importing data.

Within this guide, you'll also find information about file-sharing and print spooling optimizations for production workflows and details about the many options available when running Xinet software in a production environment.

- The *Xinet Client Guide*

  This guide provides information to help end-users who are Xinet server clients—with either direct access to the server or access through a Web browser. It explains the growing list of Xinet utilities that help users get the most out of a Xinet server. It also includes tips for working within that environment when using Macintosh or Windows applications from Quark, Adobe, etc., as well as features available when sites make use of Video for Xinet, Archive for Xinet, and Xinet Portal.

- *Xinet PC Connectivity Guide*

  This guide provides information about installing, setting up and maintaining the Xinet modified Samba software which allows Microsoft Windows clients to interact with Xinet.

- The *Xinet Portal Administration Guide*

  This guide explains installation, licensing and administration for Xinet Portal sites. It also provides an overview of using Xinet Portal to customize the look and feel of user sites and gives examples of building new functionality to even further extend the product.

- The *Video for Xinet Administration Guide*

  This guide provides information about installing and configuring the Xinet Video module, including information about its various options, settings, metadata capabilities and interactions with the Xinet database.

- The *Archive for Xinet Administration Guide*

  This guide provides details about using Xinet Archive to allow Xinet users to communicate over the Web with *Symantic Backup Exec*™ archiving software. The guide explains installation and administration and provides guidelines for user interaction over the Web.

- On-line manual pages

  UNIX-style on-line *man*(1) pages are included with UNIX distributions. While not intended for the lay reader, they provide handy reference for more technically-advanced administrators and for programmers who want to work with Xinet programs from the command line. The *man*(1) pages are also included as part of each PDF manual.

- Xinet on-line help

  Context-sensitive Help buttons in the Xinet administration interfaces provide information about options in GUIs. They open appropriate pages within the *Xinet Administration Guide*.

- *Xinet TechNotes*

  Xinet TechNotes provide information about technical issues not included in manuals. Like the *Xinet Guide to Development APIs* and *man*(1) pages, the notes are often aimed at those extending Xinet products beyond functionality offered in Xinet GUIs. They also provide information on issues that change quickly or that are of a transient nature.

If you cannot solve your problems from information in the manuals, please contact your Authorized Xinet Integrator or Xinet technical support.)

Data Interchange Overview

Prior to Version 15.05 of Xinet, developers depended on a set of Xinet JavaScript CGIs for communication with its Xinet server. These included:

- *listdir* for listing directories and volumes
- *searchengine* for searching
- *toplevel* to provide an entry point for Xinet
- *imageinfo* for getting detailed image info
- *getimage* for extracting Web-ready previews of images
- *filemgr* for file management
- *basketbuttons* for determining plug-in availability
- *basketcontrol* for giving access to a user's basket
- *mview* for showing QuarkXPress, PDF and InDesign file previews
- *showbasket* for providing information about basket contents
- *streamfile* for streaming files to a browser

With Version 15.05, a single program- and platform-independent application on the Xinet server, called *portalDI,* could be used instead *(/usr/etc/webnative/portalDI* on UNIX systems and *C:\Program File\Xinet\WebNative\Bin\portalDI.exe* on Windows). The newer *portalDI* CGI, which, because of its RESTful architecture and leaner, highly-compliant JSON inquiry format, has proven to be much faster, and brings along the advantage of eliminating SOAP, which XML used to require. JSON compliance also means that the *portalDI* CGI can be used to exchange data between the Xinet server and applications written in ActionScript, C, C#, ColdFusion, Common Lisp, E, Erlang, Java, JavaScript, Lua, Objective CAM, Perl, PHP, Python, Rebol, Ruby, and Scheme or any other language that is JSON compliant.

Interaction with the *portalDI* binary is quite straightforward: your application sends CGI-compliant[1] queries for data. In turn, the *portalDI* binary on the Xinet server returns answers formatted according to the *The application/json*

1. D. Robinson and K. Coar, The Common Gateway Interface (CGI), Version 1.1, The Apache Software Foundation, October 2004 © The Internet Society, 2004, http://www.ietf.org/rfc/rfc3875. (Reprinted as "Appendix B: The Common Gateway Interface (CGI)" in this guide.)

Media Type for JavaScript Object Notation (JSON), JSON[1] RFC, with only the exact data you requested, nothing extra. (Thus, faster responses that don't require filtering.)

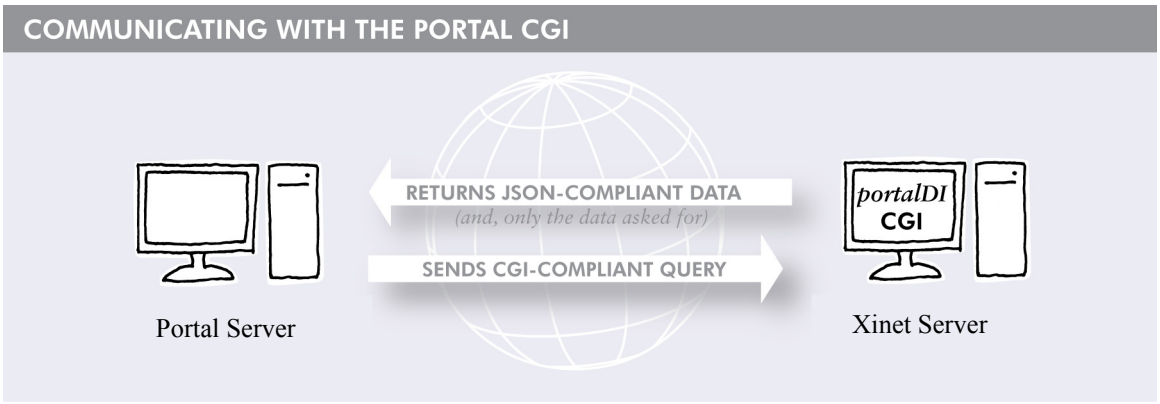

Figure 0-1 The *portalDI* CGI

For debugging use, you can use your browser's URL line to pass inquiries on to the portalDI binary and capture what the server returns using a plug-in or extension for your browser. An example follows:

Query: `portalDI?action=showbasket&showbaskbtns=true`

What it shows:

- Using `portalDI?` in the URL initiates a *portalDI* query.
- An `action=showbasket` query requests information from the `BASKET_INFO` array.
- The `showbaskbtns=true` lists available basket plug-ins and information along with the user's permission settings for them
- The reserved character `&` strings together multiple queries.

Figure 0-2 shows the query in the URL.

1. D. Crockford, *The application/json Media Type for JavaScript Object Notation (JSON)*, JSON.org, 2006 © The Internet Society, *http://tools.ietf.org/html/rfc4627.* (Reprinted as *"Appendix C: The application/json Media Type for JavaScript Object Notation (JSON)"* in this guide.)

Query the portalDI CGI *on the Xinet server.*

Show contents of the Shopping Basket (the BASKET_INFO array).

List Basket plug-ins and permissions for them

Figure 0-2 A query about the contents of the *Shopping Basket*

Figure 0-3 shows the results of the query looked at with Firebug

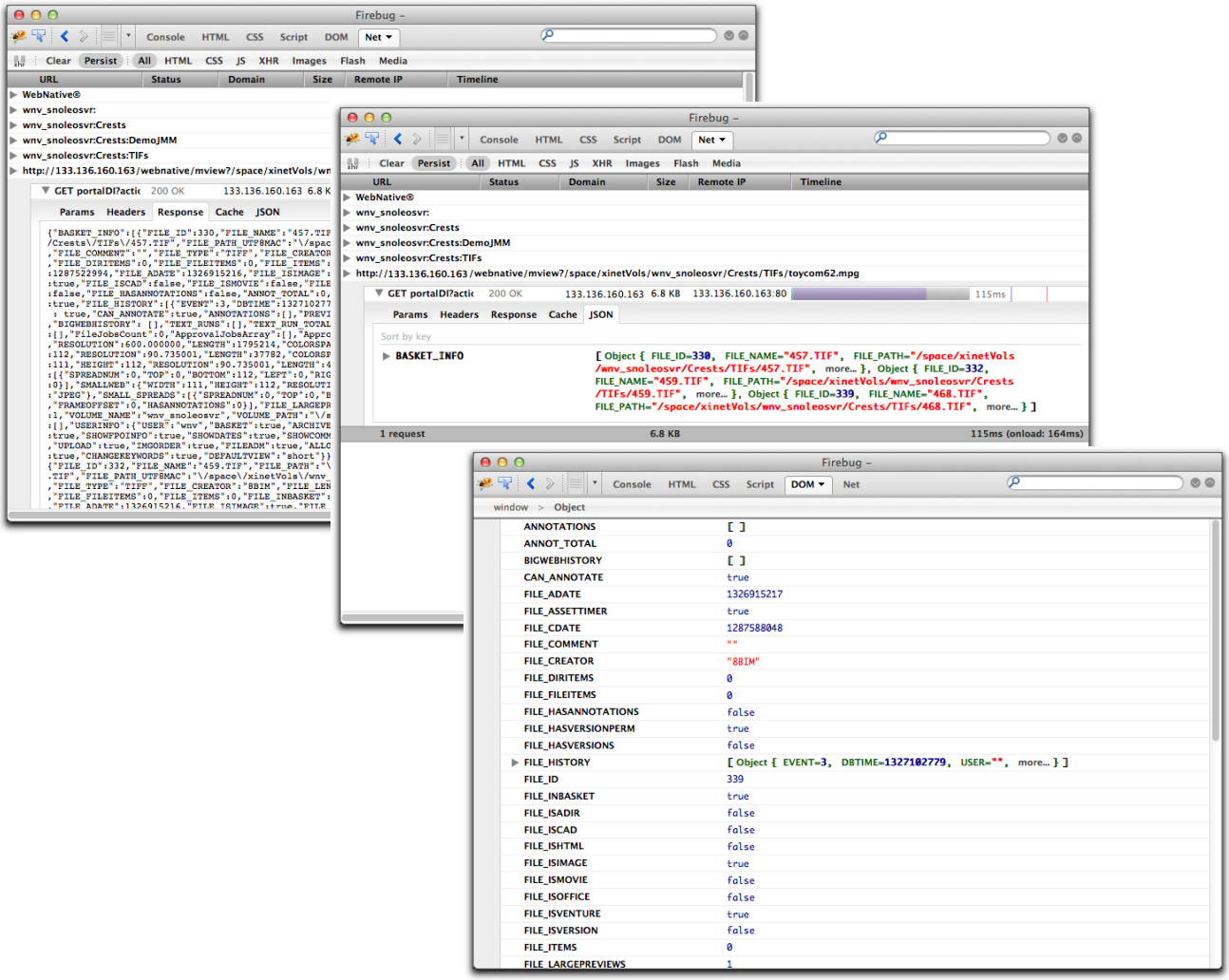

Figure 0-3 Examining the results of the query using Firebug

Syntax

3.1 Command summary

```
portalDI? [action = version | showvols | showusersettings | showkywdperms | showbaskbtns |
showiccusm | clearbasket] & [clientaddr = ipaddress] & [debug = true | false]
```

```
portalDI? [action = showdirinfo | fileinfo | showbasket | addbasket | removebasket] & [showkywds =
true | false] & [clientaddr = ipaddress] & [debug = true | false] & ( [basketname = usersbasket] |
[basketfile = /path/to/basketfile] ) & ( [fileid = fileid_num] | [path = /path/to/file] )
```

```
portalDI? [action = upload] & [dir = path/to/target/directory] & [updatedir = value] & [newdir =
new_dir_name] & [newname = file_name] & [clientaddr = ipaddress] & [overwrite = true | false]
&[create_date = unix_date] & [modification_date = unix_date] & [dir_start = value] & [dir_end =
value] & [filedata = value] & [dbkeyword = value] & [keywordN = metadata_value]
```

```
portalDI? [action = getorderimage] & [archiveformat = format] & [backcolor = value] & [blackpoint
= true | false] & [clipping = value] &[colorplate = value] &[colorspace = value] & [crop =
(cropw,croph)] & [dpi = output_resolution] & [format = value] & [height = value] & [includemeta =
true | false] & [inputicc = true | false] & [mergeclip = true | false] & [outputicc = true | false]
& [overprint =true | false] & [preview = true | false] & [pctiff = true | false] & [pict = true |
false] & [renderingintent = value] & [scale = value] & [spotcolor = true | false] & [spotoff = true
| false] & [spreadnum = true | false] & [usm = value] & [watermark = true | false] & [webready =
true | false] & [width = value]
```

```
portalDI? [action = streamfile ] & [videoid = video_num] & [filetype = value] & [attach = true |
false] & [clientaddr = ipaddress] & [debug = true | false] & ( [fileid = fileid_num] | [path = /
path/to/file] )
```

```
portalDI? [action = submitkywd] & [comment = value] & ( [keyword123 = value] & [keyword789 = value]
) & [clientaddr = ipaddress] & [debug = true | false] & ( [fileid = fileid_num] | [path = /path/to/
file] )
```

```
portalDI? [action = getimage ] & [pixels = num_pixels] & [bgcolor_r = background color red] &
[bgcolor_g = background color green] & [bgcolor_b = background color blue] & [spreadnum =
spread_num] & [filetype = value] & [archtype = value] & [archname = value] & [packerrors = true |
false] & [clientaddr = ipaddress] & [debug = true | false] & ( [fileid = fileid_num] | [path = /
path/to/file] )
```

```
portalDI? [action = filemgr ] & [filemgraction = mkdir | move | copy | delete | promote | version]
& [newpath = /path/to/destination/folder] & [newname = value] & [overwrite = true | false] &
[clientaddr = ipaddress] & [debug = true | false] &
( [fileid = fileid_num] | [filename = /path/to/original/file] )
```

```
portalDI? [action = annotations] & [page = value] & [grouponly = value] & [clientaddr = ipaddress]
& [debug = true | false] & ( [fileid = fileid_num] | [path = /path/to/file] )
```

```
portalDI? [action = saveannotations] & [page = value] & [grouponly = value] & [clientaddr =
ipaddress] & [debug = true | false] & ( [fileid = fileid_num] | [path = /path/to/file] )
```

```
portalDI? [action = browse] & [fileid = value | path = value] & [itemsperpage = value & page =
value] | [showfiles = false | [true & filesperpage = value & filepage = value] & showdirs = false
| [true & dirsperpage = value & dirpage = value]] & [resultsetid = value] & [filename_flag_i = -1
| value] & [filename_sort_i = value] & [date_sort_n_i = value ] & [dbsearch_flag_n_i = -1 | value]
& [dbsearch_sort_n_i = value] & [showkywds = true | false]
```

```
portalDI? [action = navigator] & [fileid = value | path = value] & [showfile = false | true]
```

```
portalDI? [action = presearch] & [searchid = value] & [sharesearches = true] & [delsavedsearch =
true & searchid = value]

portalDI? [action = search] &

( [subsearch_Y = value & subsearch_logic_Y = value] &

[searchall_X = value & searchall_flag_X = value & searchall_logic_X = value] &

[filename_X= value & filename_flag_X = value & filename_logic_X = value] &

[filetype_X = value & filetype_flag_X & filetype_logic_X = value] &

[date_Y_X = value & date_flag_Y_X = value & date_logic_Y_X = value] &

[comment_X = value & comment_flag_X = value & comment_logic_X = value] &

[searchallkeyword_X = value & searchallkeyword_flag_X = value & searchallkeyword_logic_X = value]
&

[searchallfti_X = value & searchallfti_flag_X = value & searchallfti_logic_X = value] &

[dbsearch_keyword_Y_X = value & dbsearch_flag_Y_X = value & dbsearch_logic_Y_X = value] &

[annotations_X = value & annotations_flag_X = value & annotations_logic_X = value] &

[highresinfo_Y_X = value & highresinfo_flag_Y_X = value & highresinfo_logic_Y_X = value] &

[videoinfo_Y_X = value & videoinfo_flag_Y_X = value & videoinfo_logic_Y_X = value] &

[filecontent_X = value & filecontent_flag_X = value & filecontent_logic_X = value] &

[event_Y_X = value & event_flag_Y_X = value & event_logic_X = value] &

[stamps_X= value & stamps_flag_X = value & stamps_logic_X = value] &

[filedir_X = value & filedir_logic_X = value] &

[filesize_X = value & filesize_flag_X = value & filesize_logic_X = value] &

[assettimer_X = value & assettimer_flag_X = value & assettimer_logic_X = value] &

quicksearch_Y_X = value & customfile_X = value & customkw_X = value ) &

[searchname = value & searchdescription = value & savesearch = true] &

[showonlinearchive = value]
```

3.2 Some *portalDI* rules for queries

The following rules, in broad strokes, govern *portalDI* queries. For complete details, please refer to "Appendix B: The Common Gateway Interface (CGI)," on page 53 and "," on page 83.

- Blank spaces

  A blank space between strings will be converted to %20, which then, becomes insignificant. Examples: `showvols=` returns the same output as `showvols =` and, `KV Photos` resolves to `KV%20Photos`, with both returning the same information.

  Other forms of blank space (horizontal tabs, Line feed or New line, and Carriage return) are also ignored when they occur before or after JSON structural characters ([ ], { }, colons, and commas).

- Reserved characters:
  “;” | “/” | “\” | “?” | “:” | “@” | “&” | “=” | “+” | “$” | “,” | “[” | “]”

  Each character string, above has special meaning and should be replaced by the corresponding “%” escape sequences when needed as a common character. The resulting string can then be used in assembling a URI.

See RFC 4627 (“Appendix C: The application/json Media Type for JavaScript Object Notation (JSON)”) for details and alternatives.

- Multiple query attributes and order

  Use `&` to separate assembled query attributes. Order doesn’t matter. The query `portalDI?fileid=1234&action=addbasket` produces the same results as the query `portalDI?action=addbasket&fileid=1234`.

- Case sensitivity

  Case sometimes matters, for example in UNIX path names it matters. Metadata *variables* are case insensitive, while metadata *values* are not, etc. Look for *case sensitivity* in the index of this volume.

- Only one `action` is allowed per query.

- Mistakes/Error handling

  In some cases an error is output. Otherwise, *toplevel* errors will be printed in the *venture* log (the *nativeadmin Logging, Database* page).

- Inquiry length

  Apache limits inquiries to 4000 characters. (Users with older Windows Internet Explorer browsers may have limitations of 2000 characters.)

3.3 WebNative *portalDI* CGI arguments

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`debug= true | false |` *value*	When set to *true, portalDI* will send Xinet debugging information to the *venture* log which can be viewed through *nativeadmin* GUI. The location of the actual log file varies according to platform. On Unix systems, it’s in */var/adm/appletalk/ venture.log*; on Windows systems, *C:\Program* Files\Xinet\FullPress\venture.log.
`clientaddr =` *ipaddress*	The *portalDI* binary will use the address specified in the Xinet event logs when populated. Otherwise, the WebNative Portal Server’s address is added to the logs.
`fileid=` *fileid_num*	An alternative to specifying a file or directory using `path =`. Used to specify a WebNative Venture *fileid* number when you want to interact with a specific file or directory. You can determine the *fileid* by searching for it in 1) the source code for a browser page displaying the asset, or 2) the source code for a WebNative *Image Info* display of the asset. Scan the code for *“FILE_ID:”*.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`path=`*/path/to/some/file*	An alternative to specifying a file or directory using `fileid =`. Used to identify a file or folder by where it resides within the Xinet file structure. Case sensitive. On both UNIX and Windows systems, use the forward slash or solidus (/) to indicate separate levels of the file system. On Windows, no need to indicate the drive (as in *C:*); Apache will determine it from configuration files. Take care when special characters are used in file names. However, as stated in RFC 4627 for JSON, any character may be escaped: "If the character is in the Basic Multilingual Plane (U+0000 through U+FFFF), then it may be represented as a six-character sequence: a reverse solidus, followed by the lowercase letter *u,* followed by four hexadecimal digits that encode the character's code point. The hexadecimal letters *A* through *F* can be upper or lowercase." There are also some two-character sequence escape representations of some frequently used characters. See *"Appendix C: The application/json Media Type for JavaScript Object Notation (JSON)," on page 83* for details.
`action = version`	When used, prints the current working version of the *portalDI* CGI. This action will also block other replies from the *portalDI* binary.
`action = showvols`	This is a bit like *toplevel*. When used, *showvols* provides a list of volumes and the current user's permissions for those volumes. This is the default action whenever the *portalDI* CGI is used without *path* or *fileid,* or whenever the *path* or *fileid* can't be found.
`action = showusersettings`	When invoked, provides basic information about the current user's assigned, non-volume-specific settings. These include *Language, Password Permissions, E-mail* address and *Group* information.
`action = showkywdperms`	When invoked, lists available keyword fields along with the user's permission settings for them.
`action = showbaskbtns`	When invoked, lists available basket plug-ins and information along with the user's permission settings for them.
`action = showiccusm`	When invoked, lists available ICC profiles and Unsharp Mask options available from the Xinet.
`action = clearbasket`	Used in conjunction with: basketname = When invoked, clears basket of its contents.Does not work if you specify a particular file for removal. Use `action = removebasket` for that. Adding a `basketname` will invoke the action on a saved basket for the current user.
`action = showbasket` Used in conjunction with: `basketname =` or `basketfile =` `showkywds = true | false`	When invoked, lists contents of the basket and their attributes. Adding a `basketname` will list the contents of a saved basket for the current user, or using `basketfile` with the full path to any user's basket will display that basket's contents. When combined with `showkywds = true` keyword values are included in output.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = streamfile` Used in conjunction with: `videoid =` *video_num* `filetype =` *value* `attach = true | false` `fileid =` or `path =`	When invoked, and used in conjunction with `fileid` or `path`, will stream the content to a user's Web browser. Content-type header is assigned when combined with `filetype`. Content-disposition set to attached when combined with `attach = true`. This will force a download action. Videos stored in the Web- Native Suite database are streamed when combined with `videoid`. Video IDs are available in a the `fileinfo` output. An ID of *0* always corresponds to the original file.
`action = fileinfo` Used in conjunction with: `showkywds = true | false` `fileid =` or `path =`	When invoked, and used in conjunction with `fileid` or `path`, will output information for a file or folder. This option is only required when detailed information is needed about folders but not necessary about files. Adding `showkywds` will add more detailed information.
`action = showdirinfo` Used in conjunction with: `fileid =` or `path =`	When invoked, and used in conjunction with `fileid` or `path`, will output information about the parent folder to the requested file/folder.
`action = addbasket` Used in conjunction with: `basketname =` `fileid =` or `path =`	When invoked, and used in conjunction with `fileid` or `path` will add the requested file/folder to the user's basket. Adding a `basketname` will invoke the action on a basket of the current user. If the basket does not exist it will be created.
`action = removebasket` Used in conjunction with: `basketname =` `fileid =` or `path =`	When invoked, and used in conjunction with `fileid` or `path` will remove the requested file/folder to the user's basket. Adding a `basketname` will invoke the action on a basket of the current user.
`action = submitkywd` Used in conjunction with: `keyword123 =` *value* `fileid =` or `path =`	When invoked, and used in conjunction with `comment =` *some comment* and a `fileid` or `path`, will change information in the file's *Show Details* field. When invoked, and used in conjunction with `keyword`*123* `=` *value* and a `fileid` or `path`, will change keyword assignments.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = getimage` Used in conjunction with: `pixels =` *num_pixels* | `[ bgcolor_r =` *background color red* `&` `bgcolor_g =` *background color green* `&` `bgcolor_b =` *background color blue* `] |` `spreadnum =` *spread_num* `filetype = small | large | high | fpo` `archtype = sit | zip | uzip | maczip | umaczip` `archname =` *value* `packerrors = true | false` `fileid =` or `path =`	When invoked, and used in conjunction with `fileid` or `path` and `filetype = small` or `large` or `high` or `fpo`, will send output to the user's Web browser. Small or large requests provide previews in JPEG or GIF formats depending on how they are stored in Xinet. High or FPO requests will be sent for download as file archives. Specifying `archtype = sit` or `zip` or `uzip` or `maczip` or `umaczip` will define the archive format. Additionally an alternative name for the archive can be requested with `archname =` *value*. Specifying `pixels =` *num pixels* will display a preview of the selected image at a specific size. Padding of the image is automatic. The backgound color of the pad is defined with `bgcolor_r`, `bgcolor_g`, and `bgcolor_b` using values *0* to *255*.
`action = filemgr` Used in conjunction with: `filemgraction = mkdir move | copy | delete | promote | version` `newpath =` */path/to/destination/folder* `newname =` *value* `overwrite = true | false` `fileid =` or `filename =`	When invoked, and used in conjunction with `fileid` or `filename` and `filemgraction =` *option,* will preform the requested action on the Xinet server. The `delete` option only requires a `fileid` or `filename` be given for the action. The `mkdir` action will create a new folder on the Xinet server in the `newpath` with the name specified as `newname`. Actions `move`, `copy` and `promote` require an originating `fileid` or `filename` and a destination `newpath` and `newname`. `Overwrite = true` will cause the action to replace any existing file or folder in the `newpath` with matching `newname`.
`action = annotations` Used in conjunction with: `page =` *value* `grouponly =` *value* `fileid =` or `path =`	When invoked, lists all active annotations for the specified page in the given file. A `page` number and `path` or `fileid` must be specified. Page indexing begins at zero.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = saveannotations` Used in conjunction with: `fileid =` or `path =`	When invoked, saves the supplied arguments as an annotation for the specified page in the given file. A page number and path must be specified. Annotation arguments must be supplied in a POST request. Annotations are specified with the following arguments: `save_annotations =` A comma-delimited list of annotation IDs. Each entry in the list should provide annotation details to save, as described below. NOTE: The *N* in each argument name represents the annotation ID of the annotation to which the argument corresponds. For a new annotation, use any unique string. • *N*`page =` Page to which the annotation applies • *N*`position`*X* `=` The *X* position of the left side of the annotation, in pixels. • *N*`position`*Y* `=` The *Y* position of the top side of the annotation, in pixels. • *N*`dimension`*H* `=` The height of the annotation, in pixels. • *N*`dimension`*W* = The width of the annotation, in pixels. • *N*`notes =` Notes for the annotation. • *N*`visible =` Indicates whether the annotation is visible. Can be `true` or `false`. • *N*`zOrder =` The *z*-index of the annotation. Can be any integer. • *N*`zoom =` The zoom level at which the annotation was made. Can be any real number. • *N*`type =` The type of the annotation. Can be `Rect`, `Text`, `Stamp`, or `Sketch`. If the annotation is type `Text`, the following arguments should be supplied: • *N*`contents =` The body of the text annotation. • *N*`size =` The size of the text, in points. • *N*`font =` The font family used to display the annotation. Can be `Sans-Serif`, `Serif`, `Monospace`, `Cursive`, or `Fantasy`. • *N*`color =` The color used to display the annotation. Should be an integer. Hex values in *0xN* notation are fine. • *N*`italic =` Indicates whether to use italics for the annotation. Can be `true` or `false`. • *N*`bold =` Indicates whether the annotation should be bold. Can be `true` or `false`.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = saveannotations` (continued)	• *N*`justification =` How to justify the annotation. Can be `left`, `right`, or `center`. If the annotation is type `Rect`, the following arguments should be supplied: • *N*`width =` Border width for the annotation, in pixels. • *N*`style =` Border style for the annotation. Can be `solid`, `dashed`, `beveled`, or `inset`. • *N*`fill =` Fill color for the annotation. Should be an integer. Hex values in *0xN* notation are fine. • *N*`stroke =` Stroke color for the annotation. Should be an integer. Hex values in *0xN* notation are fine. If the annotation is type `Stamp`, the following arguments should be supplied: *N*`stamp =` Stamp index for the annotation. Should be an integer. Indices correspond as follows: `0` = *Approved* `1` = *Confidential* `2` = *Departmental* `3` = *Draft* `4` = *Experimental* `5` = *Expired* `6` = *Final* `7` = *For Comment* `8` = *For Public Release* `9` = *Not Approved* `10` = *Not For Public Release* `11` = *Sold* `12` = *Top Secret* If the annotation is type `Sketch`, the following arguments should be supplied: • *N*`width =` Stroke width for the annotation, in pixels. • *N*`color =` Stroke color for the annotation. Should be an integer. Hex values in *0xN* notation are fine. • *N*`path =` Stroke path for the annotation. Should be a colon-delimited list of *X,Y* pairs. For example, a five by five square with an origin at 1,1 could be encoded as `1,1:6,1:6,6:1,6:1,1.`

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = browse` Used in conjunction with: `[ itemsperpage =` *value* `page =` *value* `] |` `[showfiles = false |` `[ true filesperpage =` *value* `filepage =` *value* `]` `showdirs = false |` `[ true dirsperpage =` *value* `dirpage =` *value* `]]` `resultsetid =` *value* `filename_flag_`*i* `= -1 |` *value* `filename_sort_`*i* `=` *value* `date_flag_n_`*i* `= -1 |` *value* `date_sort_n_`*i* `= value` `dbsearch_flag_n_`*i* `= -1 |` *value* `dbsearch_sort_n_`*i* `=` value `showkywds = true | false`	When invoked with a folder's *fileid* or a folder's *full path,* will list the contents of the given folder. Some specific arguments: • `itemsperpage` = number of files and folders to list per request. When more items exist then defined for `itemserpage` the results will be paginated. • `page` = value of page number to be shown. Not to be used with above arguments: • `showfiles` = request specific output of files. • `filesperpage` = number of files to list per request. When more items exist then defined for `filesperpage` the results will be paginated. • `filepage` = value of file page number to be shown. • `showdirs` = request specific output of folders. • `dirsperpage` = number of folders to list per request. When more items exist then defined for `dirsperpage`, the results will be paginated. • `dirpage` = value of directory page number to be shown. Additional arguments: • `resultsetid` = pervious request result set from the Search Engine. • `filename_flag_`*i* = when submitted with `-1` allows the Search Engine to list results with a specified sort order. • `filename_sort_`*i* = sets Search Engine sort flag for `filename` and its sort order. • `date_flag_n_`*i* = when submitted with `-1` allows the Search Engine to list results with a specified sort order. *n* defines date type. *n* = `4` for *access date* and *n* = `3` for *modified date*. • `date_sort_n_`*i* = sets the Search Engine sort flag for dates and its sort order. *n* defines *date type*. *n* = `4` for access date and *n* = `3` for *modified date*. • `dbsearch_flag_n_`*i* = when submitted with `-1` allows the Search Engine to list results with a specified sort order. *n* defines *keyword field id*. • `dbsearch_sort_n_`*i*: Sets the Search Engine sort flag for keyword field n and its sort order. n defines keyword field id. • `showkywds` = when *true* retrieves *keyword data* for each file and or folder listed.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = navigator` Used in conjunction with: `fileid =` *value* | `path =` *value* `showfiles = false | true`	When invoked with a list of folders' *fileids* or folders' *full paths*, will output an ordered directory tree. Some specific arguments: `showfiles =` when `true`, will include files at each folder depth requested. Example: `http://172.16.0.20/` webnative/portalDI?action=navigator&fileid=6047&fileid=6044&showfiles=true
`action = upload` Used in conjunction with: `path =` */path/to/target/dir*	To transfer a file to the Xinet server, a client should POST requests to a URI of the following form: `portalDI?action=upload&path=`*/path/to/target/*directory Note: portalDI expects arguments to be encoded in application/x-www-form-urlencoded. When uploading files, however, the client must encode the request arguments in multipart/form-data. Most developers should never have to worry about the details of the encoding format; they should be handled by whatever HTTP library or utility is being used. Should it become necessary to encode the MIME stream by hand, details can be found at the following address: http://www.w3.org/TR/html401/interact/ forms.html#h-17.13.4.2 The cURL command-line utility is a particularly easy way to transfer files to the Xinet server. A sample invocation is show below. Elements in italics should be replaced with values appropriate to the server. `curl --verbose -u "`*username:password*`" \` -F 'overwrite=true' \ -F 'action=upload' \ -F 'keywordXX=cURLed' \ `-F 'dir=`*/server/path/to/target/dir*`' \` -F 'filedata= *@/Local/path/file/to/upload.jpg*`' \` `'http://`*hostname*`/webnative/portalDI?` `action=upload&path=`*/server/path/to /target/dir*`' \` Some specific arguments: • `dir =` */path/to/target/directory* The target directory in which uploaded files should be placed on the Xinet server. • `newdir =` *new_directory_name* If this argument is present in the MIME stream, then portalDI will create a new directory in the target directory `dir`. Its name will be the same as the argument's value. • `newname =` *new_file_name* This value will be used to name the file, instead of the filename hints made by the browser in the MIME stream.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = upload` (continued)	• `clientaddr =` *ipaddress* Host address of the client where the file originated. • `overwrite` = *true* | *false* If this argument has a value of "true" or "1," and a file with the same name already exists on the server, then the version on the server will be overwritten by the version being uploaded. • `create_date =` *unix_date* If present, then portalDI will adjust the uploaded file's creation date to match the value. The value should be a valid unix-style timestamp. • `modification_date` = *unix_date* If present, then portalDI will adjust the uploaded file's modification date to match the value. The value should be a valid unix-style timestamp. • `dir_start` = *true* | *false* If given, portalDI will instruct dblogd to suspend upload event processing until the dir_end argument is given in a later transaction. This argument is useful when uploading directory contents, as uploads are often moved by triggers/actions from a dropbox to a more appropriate location. With dir_start/dir_end, triggers won't fire until the client has transferred every file. • `dir_end` = *true* | *false* When found, portalDI instructs dblogd to resume upload event processing. If dir_start was never provided by the client, it has no effect. • `filedata` = *value* The actual file data to be stored on the server. • `keyword`*N* = *true* | *false* If present, the value of this argument will be applied to the metadata field with ID "N" for the uploaded file. The user must have permission to access to thegiven field for the value to take effect.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action= getorderimage` Used in conjunction with: `path =` */path/to/target/dir*	Some specific arguments: • `archiveformat = value` If the `webready` argument is not supplied, the resulting image will be stored in an archive file. This argument determines which format portalDI should use. If it's omitted, an appropriate default will be used. The following values are accepted: zip = Plain zip file unzip = Uncompressed zip file maczip = Zip file with finder info and resource fork umaczip = Uncompressed zip file with finder info and resource fork. • `backcolor = value` Sets the background color, when a Masked or Clipped image is output in a format that does not support masks or Clipping paths to the given color. • `blackpoint = true | false` Enables black point compensation during ICC color correction. • `clipping = value` Determines how clipping paths and masks are handled during conversion. The following values are accepted: *[empty] = Ignore clipping paths and masks* *2 = Ignore only clipping paths* *3 = Causes all non-pixel data to be omitted from the output image.* *4 = Forces pixels to be clipped to a selected clipping path even if the output format supports clipping paths.* *5 = Same effect as empty string.* • `colorplate = value` Select a color plate from the input image to render in greyscale. The following values are accepted: *C = Cyan Plate* *M = Magenta Plate* *Y = Yellow Plate* *K = Black Plate* • `colorspace = value` Process the output to a colorspace. The following values are accepted: *Grey, RGB, LAB, and CMYK*

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action= getorderimage` (continued)	• `crop =` *(cropw, croph)* The argument specifies a rectangle on the source image, in pixels, numbered from (0,0), which is the pixel in the upper-left corner of the image. The coordinates cropx (horizontal) and cropy (vertical) provide the coordinates of the first pixel that will be placed in the upper-left corner of the output image. The pair, cropw (width) and croph (height) give the number of pixels of the original image to include in the cropped image. • `dpi =` *value* Output resolution to DPI (dots per inch) • `format =` *value* File format for output. Those formats marked with * all take an optional number following the fomat (e.g., -x eps83). This number sets a JPEG compression factor (normally off for EPS format). The JPEG compression factor is a number with a useful range between 5 and 95, where 5 creates a highly compressed image that doesn't look very much like the original, and 95 produces a less-compressed image, which will look very much (but not exactly) like the original. When JPEG output format is requested, the default compression factor is 75. Requesting EPS format with a compression value of 0 will cause the output to be HEX encoded (instead of the default BINARY); but not JPEG compressed.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = getorderimage` (continued)	• `format =` *value* (continued) The following values are accepted: *eps* - Encapsulated PostScript* *tif - Tag Image File Format* *gif - Composure's Graphics Interchange Format* *jpg* - Native JPEG, "lossy" compression format* *web* - GIF (if image is masked/clipped) or JPEG* *bmp - MS Windows BitMaP (aka DIB) format* *png - Portable Network Graphic Format* • `height =` *value* Scale to a specific height. • `includemeta =` *true* | *false* Write the file's XMP data to the output image, if the output format supports it. • `mergeclip =` *true* | *false* Forces pixels to be clipped to a selected clipping path even if the output format supports clipping paths. • `outputicc =` *true* | *false* Sets an output ICC profile pathname. This option enables ICC color correction if one of the following conditions is met (precedence in the order listed): *- An input profile has been supplied with the inputicc option* *- The image has an embedded profile* *- A default profile has been specified on the Xinet server for the image format and colorspace in the file* `/var/adm/appletalk/coloropts` *[UNIX] or* C:\Program Files\Xinet\Full- `Press\Admin\coloropts` *[WINDOWS]. Note:* Xinet creates/updates the coloropts file whenever the administrator saves default ICC profiles in a Print Queue for various formats). *- The input image is in LAB colorspace.* • `overprint =` *true* | *false* Causes any file that must be rendered (PDFs and vector EPS, for example) to be rendered in CMYK, regardless of what output colorspace was requested. This option guarantees that Overprints in the image are preserved.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = getorderimage` (continued)	• `preview =` *true* | *false* This option causes imageorder to search for a "preview" image before reading in the "full-scale" image. • `pctiff =` *true* | *false* If an EPS image has a PC-style TIFF preview, this option causes the TIFF to be read instead of the EPS. Alternatively, the argument can be 1 to 4 comma-separated numbers giving the parameters for the filter. The sharpening parameters include: The radius, in pixels, of the blurring Aperture, which must be greater than 0.5 (and has no default) A percentage strength of the sharpening values to apply (default is 100), where anything 0 or less disables the Unsharp Mask filter A lower threshold, in pixel intensity (0-255), below which no sharpening will occur (default is 0). An upper limit on the sharpening that will be applied to the image (default is 255, and any number out of the 1-255 range will be ignored). A value of 255 means no limit, a value of 10 means the maximum amount of change to a pixel to 10 (in component intensity value units, which are 0-255). Note that the Unsharp Mask parameters differ from Photoshop in the radius only: add one to a Photoshop Unsharp Mask Radius to get the equivalent sharpening. • `watermark =` *true* | *false* Turns on watermarking for the output image. This turns on automatically if the Xinet installation has not been licensed. By default, a watermark is tiled, pixel-for-pixel across and down the image as many times as it will fit. • `webready =` *true* | *false* Output is ready for display in a web page from an <img> HTML tag, rather than stored in an archive format. • `width =` *value* Scale to a specific width
`action = presearch`	Returns the following arrays: *SAVEDSEARCHES_INFO*: Contains the list of saved Searches for this user *STAMPS_INFO*: Contains the stamps that can be searched on. *TYPEGROUPS_INFO*: Contains the list of file types used to generate the *File Type* Search filter. Optionally used with: • `searchid =` *value* Used to load the filters of a saved Search. • `sharesearches =` *true* Also loads saved group Searches, which are prefixed with an asterix • `delsavedsearh = true & searchid =` *value* Used to delete a saved Search.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`action = search`	For some examples of *portalDI Search* filters, see "Search filter examples," on page 37. *About filter*`_logic` Each *Search* filter can include a logic value, e.g., `searchall_logic_`*X* = *value* and `filename_- logic_`*X* = *value*. Possible values include: `1` = *AND* `2` = *OR* `3` = *XOR* If no logic value is given, then an *AND* is assumed, which is typically what most people will use.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
About filter`_flags`	`_flag` can take on the following values: `0` = *Is null* `1` = *Contains* `2` = *Start With (text) / After (dates) / Greater than* (numbers) `3` = *Ends With (text) / Before (dates) / Lesser than* (numbers) `4` = *Is Exactly (text) / On (dates)* `5` = *Has Any* `6` = *Has Each* `7` = *Contains the word* `8` = *Regular Expression* `9` = *Aggregate Has Each* `10` = *In the last (dates)* `11` = *In the next (dates)* `100` = *Is not null* `101` = *Does not contain* `102` = *Does not start with (text) / Not before (dates) /* Not greater than (numbers) `103` = *Does not end with (text) / Not after (dates) / Not* lesser than (numbers) `104` = *Is not exactly (text) / Not on (dates)* `105` = *Does not have any* `106` = *Does not have each* `107` = *Does not contain the word* `108` = *Not regular expression*
About _X and _Y_X	A few *filter*`_flags` have special requirements. The *_X* is the index of the filter. It is used to differenciate multiple filters of the same type in a *Search*. For example, if one wanted to search for *filename contains mango*, but also *filename ends with pdf*, one would construct the query as: filename_0=mango & filename_flag_0=1 & filename_1=pdf & filename_flag_1=3 The *_Y* is different. Certain types of filters have an extra value that needs to be set. For example, single *keyword Searches* need to pass the *keyword id* of the *metadata Field* upon which to perform the *Search*. *Date* filters need to specify which of the 4 *Date Fields* to *Search* upon when the *Search* is executed. The *Y* value is used for these purposes.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
Search Filters:	Where *Y* is the *SearchID* of the previous executed Search, and *X* is the *ResultSetID* of the previous executed Search. This filter request a `subsearch_logic_ Y` value.
`subsearch_Y=X`	Subsearching is meant to be used with the WebNative Portal *Quick Search* filter functionality. Essentially, you provide the Search Engine with the *SearchID* and *ResultSetID* of a previously-executed Search filter, and then add other Search filters to the Search. The Search Engine will use the previously cached Search results and apply the new filters to those results. Cached results are refreshed automatically when needed.
`searchall_X` `searchall_flag_X` `searchall_logic_X`	Searches file name, Comments, text content, Annotations and text-based metadata fields assigned to the user's template. An optional `searchall_flag_X` value can be passed. If missing, a value of `1` (*contains*) is assumed.
`filename_X` `filename_flag_X` `filename_logic_X`	Adds a file name filter. Requires a `filename_flag_X` value to be passed as well.
`filetype_X` `filetype_flag_X` `filetype_logic_X`	Adds a file type filter. The value of this parameter corresponds to the *TypeGroupID* column of the *searchtypegroups* table. A `filetype_flag_X` should always be `4` (*Is Exactly*). Even if set to another value, it will be changed to `4`. Still, it is important to provide a value because the absence of a `filtype_flag_X` value tells the *Search Engine* that it has encountered a legacy file type *Search,* which won't work with the new values. So, always pass `filetype_flag_X=4` along with `filetype_X` filters.
`date_Y_X` `date_flag_Y_X` `date_logic_Y_X`	Adds a date filter. Requires a `date_flag_Y_X` value to be passed as well. The value of *Y* determines which date field is searched: `1` = *Create date* `2` = *Backup date* `3` = *Modify date* *Anything else* = *Access date* The value of this parameter should be in UNIX timestamp format.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`comment_flag_x` `comment_logic_X`	Adds a *Finder Comment* filter. Requires a `comment_flag_X` value to be passed as well.
`searchallkeyword_X` `searchallkeyword_flag_X` `searchallkeyword_logic_X`	Searches all text-based metadata Fields assigned to the user's template but not necessarily indexed in the *wnvfti* indexes. Requires a `searchallkeyword_flag_X` value to be passed as well.
`searchallfti_X` `searchallfti_flag_X` `searchallfti_logic_X`	Searches all text-based metadata Fields assigned to the user's template and indexed in *wnvfti* indexes. Requires a `searchallfti_flag_X` value to be passed as well.
`dbsearch_keyword_Y_X` `dbsearch_flag_Y_X` `dbsearch_logic_Y_X`	Adds a filter for *Keyword ID Y*. Requires a `dbsearch_flag_Y_X` value to be passed as well.
`annotations_X` `annotations_flag_X` `annotations_logic_X`	Adds a filter for *Annotations* Notes and Comments. Requires a `annotations_flag_X` value to be passed as well.
`highresinfo_Y_X` `highresinfo_flag_Y_X` `highresinfo_logic_Y_X`	Adds a *highresinfo* filter, where *Y* is: `1` for *width* `2` for *height* `3` for *resolution* `4` for *colorspace* `5` for *width* OR *height* `6` for *width* AND *height* `7` for *ICC profile* Requires a `highresinfo_flag_Y_X` value to be passed as well. For *colorspace*, the value can be: `1` for *mask* `2` for *1-bit* `4` for *grayscale* `8` for *indexed colors* `16` for *RGB colors* `32` for *CMYK colors* `64` for *Lab colors* `128` for *RGB high* You can add `1` to `4`, `16`, `32` and `64` to combine a *colorspace* and the *has mask* option.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`videoinfo_`*Y*`_`*X* `videoinfo_flag_`*Y*`_`*X* `videoinfo_logic_`*Y*`_`*X*	Adds a *videoinfo* filter, where *Y* is: `2` for *video format* `3` for *codec name* `4` for *width* `5` for *height* `6` for *duration* `7` for *bitrate* `8` for *width* OR *height* `9` for *width* AND *height* Requires a `video_flag_`*Y*`_`*X* value to be passed as well.
`filecontent_`X `filecontent_flag_`*X* `filecontent_logic_`*X*	Adds a *file content* filter. Requires a `filecontent_-flag_`*X* value to be passed as well.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`event_Y_X` `event_flag_Y_X` `event_logic_Y_X`	Adds an event filter, where *Y* is: `3` for *CREATE* `4` for *DELETE* `5` for *RENAME* `6` for *READ* `7` for *WRITE* `8` for *COPY* `9` for *MKDIR* `10` for *RMDIR* `11` for *ADDCMT* `12` for *SETPARAM* `13` for *FPO* `14` for *WEBIMAGE* `15` for *QUARK* `16` for *PDF* `18` for *DOWNHIRES* `19` for *DOWNFPO* `20` for *DOWNPREV* `21` for *UPLOAD* `22` for *IMAGEORDER* `23` for *BACKEDUP* `24` for *ONLINE* `25` for *MEDIACHG* `26` for *TAPEDE* `27` for *PRINTED* `28` for *ARCHIVED* `29` for *DOSYNC* `30` for *TRIGGER* `31` for *METACHG* `32` for *WNA* `33` for *IMGREPL* `34` for *VIDEO* `35` for *DOWNVIDEO* `36` for *VIDVIEWED* `37` for *ANNOTNEW* `38` for *ANNOTEDIT* `39` for *ANNOTVIEW* `40` for *SETMETA* `41` for *DOXMPSYN* `43` for *ASSETSTART* `44` for *ASSETEXPIRED* `45` for *ASSETLOCKED* `46` for *ASSETUNLOCKED*
The value should be in UNIX timestamp format. Requires an `event_flag_Y_X` value to be passed as well.	

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`stamps_X` `stamps_flag_X` `stamps_logic_X`	Adds an Annotation stamp filter with the *value* being the stamp's text. Requires a `stamps_flag_X` value to be passed as well.
`filedir_X` `filedir_logic_X`	Adds a file/folder filter. A value of `0` for files only, or `1` for folders only.
`filesize_X` `filesize_flag_X` `filesize_logic_X`	Adds a file size filter. Requires a `filesize_flag_X` value to be passed as well.
`assettimer_X` `assettimer_flag_X` `assettimer_logic_X`	Adds a filter for Asset Timer status. For `assettimer_X`, possible values include: `1` = *Available* `2` = *Unavailable* `3` = *Unclassified* `assettimer_flag_X` should be set to `4`. Even if set to another value, it will be changed to `4`. `assettimer_logic_X` is optional. Defaults to `1` (*AND*).
`quicksearch_Y_X`	Runs a Quick Search with Quick Search setting *Y*. The *searchenginesettings* table should contain a `quicksearchfield_Y` and `quicksearchtype_Y` value. Otherwise the Quick Search defaults to *searchall contains*.
`customfile_X`	Adds a custom MySQL *file* table filter. No validity check is done on the content of the custom filter.
`customkw_X`	Adds a custom MySQL *keyword1* table filter. No validity check is done on the content of the custom filter.
`searchname` `searchdescription` `savesearch`	• Set the name of the saved *Search*. • Set the description of the saved *Search.* • Save the *Search* being executed.

Table 3.1 WebNative *portalDI* CGI arguments

Argument *display controls*	**What it does**
`showonlinearchive`	Set the online/archived status of files that should be returned. The *value* should be a number, using the following value bitmap: `1` Use *AND* instead of *OR* between *Online* and *Archived* status `2` *Online = 0* (excludes *4*) `4` *Online = 1* `8` *Archived = 0* (exclued *16, 32,* and *64*) `16` *Archived = 1* `32` *Nearline = 1* `64` *Offline* = 1 For example, if you wanted to return files that were both *Online* and *Archived Nearline* (but not *Archived Offline*), you would use a value of `37` (`1 + 4 + 32`).

Table 3.2 *Additional file information*

`showkywds = true | false`	When *true,* includes keyword values with output of files and folders in both listings and detailed output.
`action = annotations` Used in conjunction with: `page = value` `grouponly = value` `fileid =` or `path =`	When invoked, lists all active annotations for the specified page in the given file. A `page` number and `path` or `fileid` must be specified. Page indexing begins at zero.

Use:

`showastxt = true | false`
`s`
`howlinks`, `showhistory` and `showversions` arguments, previously used with
`action =` `fileinfo`

Table 3.2 *Additional file information*

The following, used in conjunction with `action=search`
`searchaction = search | more | less | clear`
`pathtype = value`
`pathtypelogic = value`
`selectedvol = value`
`selectedvollogic = value`
`comment = value`
`commentlogic = value`
`commenthasany = value`
`filenamelogic_i = value`
`filetypelogic_i = value`
`date_i = value`
`datelogic_i = value`
`dbsearchalllogic = value`
`custquery = value`
`dbsearchkeyword123_i = value`
`dbsearchlogic123_i = value`
`dbsearchflag123_i = value`
`dbsearchtype123_i = value`
`maxmatches = value`
`maxtotalmatches = value`
`pathsearchtype_i = value`
`skipcount = value`
`rowoffset = value`
`rowoffsetcount = value`
`rowoffsetarray = value`
`dateon_i = value`
`datewhence_i = value`
`year_i = value`
`month_i = value`
`day_i = value`
`exactmatch = value`
`nocasematch = value`
`casematch = value`
`skiphidden = true | false`
`dbsearchallkeyword = value`
`dbsearchallwithfile = value`
`dbsearchallwithcomment = value`
`filecount=integer`

Table 3.2 *Additional file information*

The following, used in conjunction with `action=search` (*continued*)
`filestart=`*integer*
`dircount=`*integer*
`dirstart=`*integer*
`joindirfile= true | false`
`treedepth=`*integer*
`showfiles = true | false`
`showlinks = true | false`
`showhistory = true | false`
`showversions = true | false`

3.4 Search filter examples

These examples show examples of *portalDI Search* filters.

- [*Search All*] [*Contains*] [*mango*]

  ```
  http://127.0.0.1/webnative/portalDI?action=search&
  searchall_0=mango&searchall_flag_0=1
  ```

  In this simple example, *portalDI* passes the *Search* value in the *searchall* variable with index `0` and the *searchall* flag for that same index. Here, `searchall_flag_0=1` could be left out since the default flag value for a searchall is `1` (*contains*).

- [*Filename*] [*Starts With*] [*mango*]

  ```
  http://127.0.0.1/webnative/portalDI?action=search&
  filename_0=mango&filename_flag_0=2
  ```

  This example uses a *filename* filter and sets the flag to `2` for [*starts with*].

- [*Create Date*] [*Before*] [*2012-02-25 13:24:56*] [*And*] [*Filename*] [*Has Each*] [*mango pdf*]

  ```
  http://127.0.0.1/webnative/portalDI?action=search&
  date_1_0=1330194296&date_flag_1_0=3&filename_0=mango pdf&
  filename_flag_0=6&filename_logic_0=1
  ```

  A little more complex, this example uses two filters. The first one is on the *Create Date* where the value, `1330194296`, is the UNIX timestamp that represents *2012-02-25 13:24:56*. The second filter is on the filename and is a token *Search* looking for both *mango* and *pdf* to be part of the filename. Note that the `filename_logic_0=1` part could have been left out since the default logic value between filters is `1` (*and*).

- [*Field 128*] [*Is Greater Than*] [*0*]

  ```
  http://127.0.0.1/webnative/portalDI?action=search&
  keyword_128_0=0&keyword_flag_128_0=2&maxfiles=10&keyword_sort_128_0=-1
  ```

  This example shows a keyword *Search* on a metadata field for values that are higher than `0`, but with the results sorted in reverse order of those values (`keyword_sort_128_0=-1`) and it also limits the results to the first 10 (`maxfiles=10`). Essentially this *Search* finds the files that have the 10 highest values for *keyword ID 128*.

Retrieving Information PHP Example

4.1 Retrieving information and assigning it to an array

These two examples, the first using GET, the second POST, show how to use *portalDI* via PHP to retrieve information and assign it to an array.

Using GET

```
$get_string = "action=showkywdperms";


$POST = "GET /webnative/portalDI?$get_string HTTP/1.0\r\n";
#must be HTTP/1.0; can't use HTTP/1.1


$POST .= "HOST:" . $host . ":" . $port . "\r\n"
. "Authorization: Basic " . base64_encode("$username:$password") . "\r\n"


#. "Content-type: application/x-www-form-urlencoded\r\n"
# Content-type only needed for post


#. "Content-length: " . strlen($request) . "\r\n"
# Content-length only needed for post


. "Connection: close\r\n\r\n";
# must always have two line endings at the end


$fp = @fsockopen($host,$port,$errno,$errstr,300);


# read buffer, excluding any headers that we don't need
if ($fp) {
fwrite($fp, $POST . $request);
$unneeded = array('www-authenticate','http','server','date', 'connection' );
do {
$skipped = false;
$header = fgets ( $fp );
foreach ( $unneeded as $skip ) {
if (($skipped = stristr ( $header, $skip )) == true) {break;}
```

```
}
if ($skipped === false) {
$hds .= $header;
# header ( $header );
}
} while ( strrpos ( $hds, "\r\n\r\n" ) == false );
    while (!feof($fp)) {
        $buffer .= fgets($fp);
    }
}
fclose($fp);

# translate results from json to php array
$results_array = json_decode($buffer, true);
```

Using POST

```
<?php
$url_string = "action=showusersettings";
$form_paramaters = "field1=value1&field2=value2&field3=value3";
$POST = "POST /webnative/portalDI?$url_string HTTP/1.0\r\n";
$POST .= "HOST:" . $_SESSION[CONFIG]['WNHOSTNAME'] . "\r\n"
. "Authorization: Basic " . $_SESSION["USER"] . "\r\n"
. "Content-type: application/x-www-form-urlencoded\r\n"
. "Content-length: " . strlen($form_paramaters) . "\r\n"
. "Connection: close\r\n\r\n";
$fp = @fsockopen($host, $port, $errno, $errstr, 300);
if ($fp)
{
fwrite($fp, $POST . $form_paramaters);
$unneeded = array('www-authenticate', 'http', 'server', 'date', 'connection' );
do
{
$skipped = false;
$header = fgets( $fp );
foreach ( $unneeded as $skip )
{
if (($skipped = stristr( $header, $skip )) == true)
{break;}
}
```

```
if ($skipped === false)
{
$hds .= $header;
}
} while ( strrpos( $hds, "\r\n\r\n" ) == false );
while (!feof($fp))
{
$buffer .= fgets($fp);
}
}
fclose($fp);

$results_array = json_decode($buffer, true);
?>
```

Archiving and Restoring Files

5.1 Archiving Files

To set a file as being archived, the database must have a record of the archive in the ArchiveFile table and a corresponding entry in the ArchiveMedia table to which the `archivefile` entry is associated. To accomplish this, `venturelog -mediachg` is used to create a media entry, and `venturelog -archived` is used to set the file as archived. Finally, once the file has successfully been archived, it should be deleted from the archived location on the file system.

Note:

`venturelog` binary is found in the following directories:

- On Unix: `/usr/etc/venture/bin/`
- On Windows: `Program Files (x86)/Xinet/Venture/bin/`

`ksmv`, `kscp` and `ksrm` commands are found in the following directories:

- On Unix: `/usr/etc/appletalk/`
- On Windows: `Program Files (x86)/Xinet/FullPress/`

Archive Tables

There are several tables that store information about archives.

- ArchiveMedia Table: Stores archive media information.

  The ArchiveMedia table stores the names of each piece of media used in the changer, as well as its state. Nearline means in the drive and Offline means not in the changer. Nearline state encompasses all interactions with the third-party tools. In order to keep this table up-to-date, the `venturelog -mediach` event is used to update the information.

- ArchiveFile Table: Stores information about archived files.

  The ArchiveFile table contain entries for archived files and folders. Each of these items also has a corresponding entry in the file and path tables which is used as a reference to these entries. For example, ArchiveFile entries are cross-referenced by the file tables FileID.

  Essentially, when a file is archived or a sync gathers information on archived files, that information is reported to the database. First, the media name is recorded (if new) or verified (if existing). Then the information about the file that is being archived or backed-up is inserted into the appropriate table. When browsing a Xinet volume, the files and folders that are in the path being displayed come from the database. If an archived file is in a path, it will be shown when browsing along with any live files that may exist there too.

To archive files:

1. Create an archivemedia entry.

   ```
   venturelog -u UserName -fn -mediachg MyMediaName nearline=100 LocationInformation
   ```

Note: The `nearline` flag value is set. This value must be a known value and must be coordinated with Xinet to function properly. The location information is an arbitrary string that is associated with the media and its current state.

2 Set a file as being archived.

```
venturelog  -u UserName -fn -archived /path/to/archived/file_archive isfile -m MyMediaName
-an 123456 -ad 1234567890
```

Note: The media name must match the previously set in the `mediachg` event above. The archive number (`-an`) is an arbitrary number set by the archive solution. The archive date (`-ad`) is a Unix time stamp of the archive time.

3 Delete the file once the file has been archived.

```
ksrm /path/to/archive/file_to_archive
```

5.2 Restoring Files

Restoring an archived file needs the file to be restored to the original location to have it be reconnected to previously generated previews and stored metadata. Calling the venturelog -`online` flag on the file indicates to the Venture database to set the file table `Online` flag to 1. Restoring to an alternate location should still first return the file to the original archived location before moving the file to the desired destination folder.

To restore a file:

1 "Restore" the archived file to the original archived location.

Note: The `-v` flag with the `ksmv` is used to prevent venture event to be generated.

```
ksmv -v /temp/source/location/file_to_restore /path/to/archived/file_archive
```

2 Set the archived file as been restored.

```
venturelog -u UserName -fn -online /path/to/archived/file_archive
```

3 **Optional**: Restore the archived file to an alternate location.

Run both of the following commands:

Copy:

```
kscp /path/to/archived/file_archive /path/to/alternate_location
```

Remove:

```
ksrm -r /path/to/archived/file_archive
```

Further reading

- Crockford, D., “The application/json Media Type for JavaScript Object Notation (JSON),” JSON.org, © The Internet Society, http://tools.ietf.org/html/rfc4627, 2006. (Included as *“Appendix C: The application/json Media Type for JavaScript Object Notation (JSON)” beginning on page 83*.)
- Gregorio, Joe, “How to create a REST Protocol,” *http://bitworking.org/news/How_to_create_a_REST_Protocol*, 2006.
- Robinson, D., Coar, K., “The Common Gateway Interface (CGI) Version 1.1,” The Apache Software Foundation, © The Internet Society, 2004, *http://www.ietf.org/rfc/rfc3875*, October 2004. See “Appendix B: The Common Gateway Interface (CGI)” beginning on page 53.
- Udell, Jon, “The Beauty of REST,” *http://www.xml.com/pub/a/2004/03/17/udell.html,* 2004.
- Udell, Jon, “Tangled in Threads: The power of the URL-line,” *http://207.22.26.166/bytecols/2001-08-15.html,* 2001.

Appendix A: Command Line Manual Pages

This appendix contains individual descriptions of programs which Xinet installs on the server.

These programs were originally developed for the UNIX operating system, and while for the most part, the descriptions of them are accurate for Windows systems, you may occasionally find options or terminology which do not apply. Many of the file name paths may still refer to UNIX systems, e.g., /usr/etc/appletalk, /var/adm/appletalk, and /usr/etc/venture/bin . In most cases /usr/etc/appletalk maps to C:\Program Files\Xinet\FullPress , /var/adm/appletalk becomes C:\Program Files\Xinet\FullPress\Admin , and /usr/etc/venture/bin becomes C:\Program Files\Xinet\Venture\Bin.

On UNIX systems, the manual pages may be installed in the appropriate directories for access with the UNIX man (1) command.

NAME

venturelog − when the state of the filesystem changes, takes command-line arguments and sends **dblogd**(1M) events

SYNOPSIS

venturelog [**−u** *username*] [**−a** *IP_Address*] [**−start** | **−cont** | **−end**] [**−fn**] *action*

DESCRIPTION

Some stages of your workflow may include custom scripts or programs that alter the state of the filesystem. It is necessary to keep WebNative Venture informed of these changes, so it can correctly display images and other files to Web clients. You could accomplish this by using **syncvoltodb**(1M) after any changes to the filesystem occur, but a full update of the system with this program would be a time-consuming process. Using **venturelog**(1M) provides a better option. The **venturelog**(1M) program takes command line arguments that send **dblog**(1M) events in the same way other Xinet programs do.

OPTIONS

−u *username*
Allows you to specify the name of web user who performed the action. This parameter is optional and its value is only shown in the detailed history tab in WebNative.

−a *IP_Address*
Allows you to specify the IP address whence the event comes. This parameter is optional and its value is only shown in the detailed history tab in the WebNative Venture GUI.

−start

−cont

−end These options are useful for **setmeta**, **trigger** and **upload** actions. They set flags on the events to create a grouped sequence of events which are to be treated together for trigger or event logging purposes (for example, when a folder full of files is uploaded).

−fn For some actions (**archived, backedup, mediachg, online** and **tapedel**), supplying **−fn** first makes older Flashnet events (as opposed to WebNative Archive events).

ACTIONS

−addcmt *filename*

Establishes that the Mac finder comment has been added or modified for *filename*.

−archived

filename **isdir** | **isfile** [**−b** *backend id*] [**−bf** *backendflags*] [**−m** *medianame*] [**−an** *archivenumber*] [**−ad** *archivedate(Unix format)*] [**−at** *thumboffset(low 32-bit)*] [**−ath** *thumboffset(high 32-bit)*] [**−al** *thumblen*] [**−info** *data*]

Establishes that *filename* has been archived. You must specify if *filename* is a file or a directory (**isfile** or **isdir**). You can optionally specify the name of the media where the archive has been made (**−m** *media name*).

−backedup *filename* **isdir** | **isfile** [**−m** *medianame*]

Establishes that *filename* is now backed up. You must specify if *filename* is a file or a directory (*isfile* or *isdir*). You can optionally specify the name of the media where the backup has been made (**−m** *media name*). This does not imply that the original in the filesystem is gone. If you are archiving the file, use the -archived flag.

−copy *filename filename2*

Establishes that *filename* has been copied to another path or filename. This event takes two filenames, source (*filename*) and target (*filename2*).

−create *filename*

Establishes that *filename* has been created in a WebNative Suite database-enabled volume; however, this will not trigger *dblogd*(1M) in such a way that a new preview for the file will be created. This flag has been preserved for historical purposes and in most cases, it is better to use *trebuild*(1M) instead.

−delete *filename*

Establishes that *filename* has been removed from a WebNative Venture enabled volume. Note: If it has been moved to the trash, use **−rename** instead. Then use **−delete** when the trash can is emptied.

−downfpo *filename*

Establishes that a user has downloaded a copy of the FPO for *filename*.

−downhires *filename*

Establishes that a user has downloaded a copy of the hires for *filename*.

–downprev *filename*
Establishes that a user has downloaded a copy of the web preview for *filename*.

-doxmpsync *filename*

Perfoms a syncxmp (-tofile) call to update the XMP data of *filename* to match the data in the database.

–fpo *filename*
Establishes that an FPO was created for *filename*.

–imageordr *filename*
Establishes that *filename* has been custom ordered from the WebNative Venture server.

–mediachg *media name* **online[=val] | nearline[=val] | offline[=val]**
Establishes that the *media name* has changed state to **online**, **nearline** or **offline**. An optional value can be set (default=1). A value of 2 indicates that the file was set by the archiveevent action.

–metachg *filename keywordid newvalue* | **NULL** *oldvalue* | **NULL**
Establishes that *filename* metadata value has been changed. This flag will not trigger actions. To trigger actions based on keyword changes, use the **–trigger** flag.

–mkdir *directoryname*
Establishes that a new directory has been created with the name *directoryname*.

–online *filename*
Establishes that *filename* has been restored from a backup/archive.

–pdfimage *filename*
Establishes that a PDF preview was created for *filename*.

–portaltrig *path to file*
Posts a Portal Trigger event for a given path, which, when the path is within the Active Path, will in turn fire the Portal Trigger Actions defined in the related Trigger Set.

–print *filename*
Establishes that *filename* has been printed.

–pubvols
Reads */var/adm/appletalk/KASPubVols* and updates the Venture volume table accordingly. (The *filename* parameter is ignored).

−quarkimage *filename*
Establishes that a Web preview was created for a QuarkXPress or InDesign document *filename*.

−read *filename*
Establishes that *filename* has been opened and closed without being written. This event is not written to the database; this flag is preserved for historical purposes only.

−rename *filename filename2*
Establishes that *filename* has been renamed or moved to another path. This event takes two filenames, source (*filename*) and target (*filename2*).

−rmdir *directoryname*
Establishes that a directory, *directoryname*, has been deleted. This event does not affect any child of the deleted directory. You will need to send delete events for each element inside the directory before sending **−rmdir** *directoryname*.

−setmeta *filename fileid keywordid new_value*
Writes a request to assign a new value to *filename*'s keyword1 Venture table entry for the keyword with id *keywordid*. The *fileid* argument is currently ignored.

−setparms *filename*
Establishes that the Mac finder parameters have been modified for *filename*, e.g., directory position, locked status, etc.

−sync *directoryname* "*syncvoltodb_parameters_in_double_quotes*"
Performs a "listdir" sync on *directoryname* using the syncvoltodb parameters passed in double quotes.

−tapedel *media_name*
Establishes that *media_name* has been discarded and all the backups in that media have been lost.

−trigger
filename fileid keywordid trigsetid unix_date newvalue oldvalue
Informs *trigsetid* that *keywordid* for *fileid* has been changed from *oldvalue* to *newvalue*. If the keyword change matched a trigger condition of *trigsetid*, this venturelog command will trigger the corresponding action of *trigsetid*.

−upload *filename*
Establishes that *filename* has been uploaded to the WebNative Venture server.

−webimage *filename*
Establishes that a WEB preview was created for *filename*.

−write *filename*
Establishes that *filename* has been opened and closed and that there was at least one write operation performed upon it.

EXAMPLES

venturelog −write path_to_mynewfile.pdf

venturelog −archived path_to_myarchivedfile isfile −m tape009

FILES

/usr/etc/venture/bin/venturelog

C:\Program Files\Xinet\Venture\Bin\venturelog.exe

Appendix B: The Common Gateway Interface (CGI)

Version 1.1

Included, courtesy of:
Network Working GroupD. Robinson
Request for Comments: 3875K. Coar
Category: InformationalThe Apache Software Foundation
October 2004

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

IESG Note

This document is not a candidate for any level of Internet Standard. The IETF disclaims any knowledge of the fitness of this document for any purpose, and in particular notes that it has not had IETF review for such things as security, congestion control or inappropriate interaction with deployed protocols. The RFC Editor has chosen to publish this document at its discretion. Readers of this document should exercise caution in evaluating its value for implementation and deployment.

Abstract

The Common Gateway Interface (CGI) is a simple interface for running external programs, software or gateways under an information server in a platform-independent manner. Currently, the supported information servers are HTTP servers.

The interface has been in use by the World-Wide Web (WWW) since 1993. This specification defines the ‘current practice’ parameters of the CGI/1.1’ interface developed and documented at the U.S. National Centre for Supercomputing

Applications. This document also defines the use of the CGI/1.1 interface on UNIX® and other, similar systems.

1. Introduction

1.1. Purpose

The Common Gateway Interface (CGI) [22] allows an HTTP [1], [4] server and a CGI script to share responsibility for responding to client requests. The client request comprises a Uniform Resource Identifier (URI) [11], a request method and various information about the request provided by the transport protocol.

The CGI defines the abstract parameters, known as meta-variables, which describe a client's request. Together with a concrete programmer interface this specifies a platform-independent interface between the script and the HTTP server.

The server is responsible for managing connection, data transfer, transport and network issues related to the client request, whereas the CGI script handles the application issues, such as data access and document processing.

1.2. Requirements

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY' and 'OPTIONAL' in this document are to be interpreted as described in BCP 14, RFC 2119 [3].

An implementation is not compliant if it fails to satisfy one or more of the 'must' requirements for the protocols it implements. An implementation that satisfies all of the 'must' and all of the 'should' requirements for its features is said to be 'unconditionally compliant'; one that satisfies all of the 'must' requirements but not all of the 'should' requirements for its features is said to be 'conditionally compliant'.

1.3. Specifications

Not all of the functions and features of the CGI are defined in the main part of this specification. The following phrases are used to describe the features that are not specified:

'system-defined'
The feature may differ between systems, but must be the same for different implementations using the same system. A system will usually identify a class of operating systems. Some systems are defined in section 7 of this document. New systems may be defined by new specifications without revision of this document.

'implementation-defined'
The behavior of the feature may vary from implementation to implementation; a particular implementation must document its behavior.

1.4. Terminology

This specification uses many terms defined in the HTTP/1.1 specification [4]; however, the following terms are used here in a sense which may not accord with their definitions in that document, or with their common meaning.

'meta-variable'
A named parameter which carries information from the server to the script. It is not necessarily a variable in the operating system's environment, although that is the most common implementation.

'script'
The software that is invoked by the server according to this interface. It need not be a standalone program, but could be a dynamically-loaded or shared library, or even a subroutine in the server. It might be a set of statements interpreted at run-time, as the term 'script' is frequently understood, but that is not a requirement and within the context of this specification the term has the broader definition stated.

'server'
The application program that invokes the script in order to service requests from the client.

2. Notational Conventions and Generic Grammar

2.1. Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [13]. Unless stated otherwise, the elements are case-sensitive. This augmented BNF contains the following constructs:

name = definition
The name of a rule and its definition are separated by the equals character ('='). White space is only significant in that continuation lines of a definition are indented.

"literal"
Double quotation marks (") surround literal text, except for a literal quotation mark, which is surrounded by angle-brackets ('<' and '>').

rule1 | rule2
Alternative rules are separated by a vertical bar ('|').

(*rule1 rule2 rule3*)
Elements enclosed in parentheses are treated as a single element.

**rule*
A rule preceded by an asterisk ('*') may have zero or more occurrences. The full form is 'n*m rule' indicating at least n and at most m occurrences of the rule. n and m are optional decimal values with default values of 0 and infinity respectively.

[*rule*]
An element enclosed in square brackets (‘[‘ and ‘]’) is optional, and is equivalent to ‘*1 rule’.

N rule
A rule preceded by a decimal number represents exactly *N* occurrences of the rule. It is equivalent to ‘*N***N rule*’.

2.2. Basic Rules

This specification uses a BNF-like grammar defined in terms of characters. Unlike many specifications which define the bytes allowed by a protocol, here each literal in the grammar corresponds to the character it represents. How these characters are represented in terms of bits and bytes within a system are either system-defined or specified in the particular context. The single exception is the rule ‘OCTET’, defined below.

The following rules are used throughout this specification to describe basic parsing constructs.

alpha = lowalpha | hialpha

lowalpha = “a” | “b” | “c” | “d” | “e” | “f” | “g” | “h” | “i” | “j” | “k” | “l” | “m” | “n” | “o” | “p” | “q” | “r” | “s” | “t” | “u” | “v” | “w” | “x” | “y” | “z”

hialpha = “A” | “B” | “C” | “D” | “E” | “F” | “G” | “H” | “I” | “J” | “K” | “L” | “M” | “N” | “O” | “P” | “Q” | “R” | “S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z”

digit = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

alphanum = alpha | digit OCTET = <any 8-bit byte>

CHAR = a lpha | digit | separator | “!” | “#” | “$” | “%” | “&” | “’” | “*” | “+” | “-” | “.” | “`” | “^” | “_” | “{“ | “|” | “}” | “~” |

CTL CTL = <any control character>

SP = <space character>

HT = <horizontal tab character>

NL = <newline>

LWSP = SP | HT | NL

separator = “(“ | “)” | “<“ | “>” | “@” | “,” | “;” | “:” | “\” | <“> | “/” | “[“ | “]” | “?” | “=” | “{“ | “}” | SP | HT

token = 1*<any CHAR except CTLs or separators>

quoted-string = <“> *qdtext <“>

qdtext = <any CHAR except <“> and CTLs but including LWSP>

TEXT = <any printable character>

Note that newline (NL) need not be a single control character, but can be a sequence of control characters. A system MAY define TEXT to be a larger set of characters than <any CHAR excluding CTLs but including LWSP>.

2.3. URL Encoding

Some variables and constructs used here are described as being ‘URL-encoded’. This encoding is described in section 2 of RFC 2396 [2]. In a URL-encoded string an escape sequence consists of a percent character (“%”) followed by two hexadecimal digits, where the two hexadecimal digits form an octet. An escape sequence represents the graphic character that has the octet as its code within the US-ASCII [9] coded character set, if it exists. Currently there is no provision within the URI syntax to identify which character set non-ASCII codes represent, so CGI handles this issue on an ad-hoc basis.

Note that some unsafe (reserved) characters may have different semantics when encoded. The definition of which characters are unsafe depends on the context; see section 2 of RFC 2396 [2], updated by RFC 2732 [7], for an authoritative treatment. These reserved characters are generally used to provide syntactic structure to the character string, for example as field separators. In all cases, the string is first processed with regard to any reserved characters present, and then the resulting data can be URL-decoded by replacing “%” escape sequences by their character values.

To encode a character string, all reserved and forbidden characters are replaced by the corresponding “%” escape sequences. The string can then be used in assembling a URI. The reserved characters will vary from context to context, but will always be drawn from this set:

reserved = “;” | “/” | “?” | “:” | “@” | “&” | “=” | “+” | “$” | “,” | “[“ | “]”

The last two characters were added by RFC 2732 [7]. In any particular context, a sub-set of these characters will be reserved; the other characters from this set MUST NOT be encoded when a string is URL-encoded in that context. Other basic rules used to describe URI syntax are:

hex = digit | “A” | “B” | “C” | “D” | “E” | “F” | “a” | “b” | “c” | “d” | “e” | “f”
escaped = “%” hex hex
unreserved = alpha | digit | mark
mark = “-” | “_” | “.” | “!” | “~” | “*” | “’” | “(“ | “)”

3. Invoking the Script

3.1. Server Responsibilities

The server acts as an application gateway. It receives the request from the client, selects a CGI script to handle the request, converts the client request to a CGI request, executes the script and converts the CGI response into a response for the client. When processing the client request, it is responsible for implementing any protocol or transport level authentication and security. The server MAY also function in a 'non-transparent' manner, modifying the request or response in order to provide some additional service, such as media type transformation or protocol reduction.

The server MUST perform translations and protocol conversions on the client request data required by this specification. Furthermore, the server retains its responsibility to the client to conform to the relevant network protocol even if the CGI script fails to conform to this specification.

If the server is applying authentication to the request, then it MUST NOT execute the script unless the request passes all defined access controls.

3.2. Script Selection

The server determines which CGI script is to be executed based on a generic-form URI supplied by the client. This URI includes a hierarchical path with components separated by "/". For any particular request, the server will identify all or a leading part of this path with an individual script, thus placing the script at a particular point in the path hierarchy. The remainder of the path, if any, is a resource or sub-resource identifier to be interpreted by the script.

Information about this split of the path is available to the script in the meta-variables, described below. Support for non-hierarchical URI schemes is outside the scope of this specification.

3.3. The Script-URI

The mapping from client request URI to choice of script is defined by the particular server implementation and its configuration. The server may allow the script to be identified with a set of several different URI path hierarchies, and therefore is permitted to replace the URI by other members of this set during processing and generation of the meta-variables. The server

1. MAY preserve the URI in the particular client request; or
2. it MAY select a canonical URI from the set of possible values for each script; or
3. it can implement any other selection of URI from the set.

From the meta-variables thus generated, a URI, the 'Script-URI', can be constructed. This MUST have the property that if the client had accessed this URI instead, then the script would have been executed with the same values for the SCRIPT_NAME, PATH_INFO and QUERY_STRING meta-variables. The Script-URI has the structure of

a generic URI as defined in section 3 of RFC 2396 [2], with the exception that object parameters and fragment identifiers are not permitted. The various components of the Script-URI are defined by some of the meta-variables (see below);

script-URI = <scheme> “://” <server-name> “:” <server-port> <script-path> <extra-path> “?” <query-string>

where <scheme> is found from SERVER_PROTOCOL, <server-name>, <server-port> and <query-string> are the values of the respective meta-variables. The SCRIPT_NAME and PATH_INFO values, URL-encoded with “;”, “=” and “?” reserved, give <script-path> and <extra-path>.

See Section 8.1, "4.1.5. PATH_INFO," on page 62 for more information about the PATH_INFO meta-variable.

The scheme and the protocol are not identical as the scheme identifies the access method in addition to the application protocol. For example, a resource accessed using Transport Layer Security (TLS) [14] would have a request URI with a scheme of *https* when using the HTTP protocol [19]. CGI/1.1 provides no generic means for the script to reconstruct this, and therefore the Script-URI as defined includes the base protocol used. However, a script MAY make use of scheme-specific meta-variables to better deduce the URI scheme.

Note that this definition also allows URIs to be constructed which would invoke the script with any permitted values for the path-info or query-string, by modifying the appropriate components.

3.4. Execution

The script is invoked in a system-defined manner. Unless specified otherwise, the file containing the script will be invoked as an executable program. The server prepares the CGI request as described in section 4; this comprises the request meta-variables (immediately available to the script on execution) and request message data. The request data need not be immediately available to the script; the script can be executed before all this data has been received by the server from the client. The response from the script is returned to the server as described in sections 5 and 6.

In the event of an error condition, the server can interrupt or terminate script execution at any time and without warning. That could occur, for example, in the event of a transport failure between the server and the client; so the script SHOULD be prepared to handle abnormal termination.

4. The CGI Request

Information about a request comes from two different sources; the request meta-variables and any associated message-body.

4.1. Request Meta-Variables

Meta-variables contain data about the request passed from the server to the script, and are accessed by the script in a system-defined manner. Meta-variables are identified by case-insensitive names; there cannot be two different variables whose names differ in case only. Here they are shown using a canonical representation of capitals plus underscore (“_”). A particular system can define a different representation.

meta-variable-name	=	“AUTH_TYPE” | “CONTENT_LENGTH” | “CONTENT_TYPE” | “GATEWAY_INTERFACE” | “PATH_INFO” | “PATH_TRANSLATED” | “QUERY_STRING” |“REMOTE_ADDR” | “REMOTE_HOST” | “REMOTE_IDENT” | “REMOTE_USER” | “REQUEST_METHOD” | “SCRIPT_NAME” | “SERVER_NAME” | “SERVER_PORT” | “SERVER_PROTOCOL” | “SERVER_SOFTWARE” | scheme | protocol-var-name | extension-var-name
protocol-var-name	=	(protocol | scheme) “_” var-name
scheme	=	alpha *(alpha | digit | “+” | “-” | “.”)
var-name	=	token
extension-var-name	=	token

Meta-variables with the same name as a scheme, and names beginning with the name of a protocol or scheme (e.g., HTTP_ACCEPT) are also defined. The number and meaning of these variables may change independently of this specification. (See also section 4.1.18.)

The server MAY set additional implementation-defined extension meta-variables, whose names SHOULD be prefixed with “X_”.

This specification does not distinguish between zero-length (NULL) values and missing values. For example, a script cannot distinguish between the two requests http://host/script and http://host/script? as in both cases the QUERY_STRING meta-variable would be NULL.

meta-variable-value = “” | 1*<TEXT, CHAR or tokens of value>

An optional meta-variable may be omitted (left unset) if its value is NULL. Meta-variable values MUST be considered case-sensitive except as noted otherwise. The representation of the characters in the meta-variables is system-defined; the server MUST convert values to that representation.

4.1.1. AUTH_TYPE

The AUTH_TYPE variable identifies any mechanism used by the server to authenticate the user. It contains a case-insensitive value defined by the client protocol or server implementation.

For HTTP, if the client request required authentication for external access, then the server MUST set the value of this variable from the ‘auth-scheme’ token in the request Authorization header field.

AUTH_TYPE	=	“” | auth-scheme
auth-scheme	=	“Basic” | “Digest” | extension-auth
extension-auth	=	token

HTTP access authentication schemes are described in RFC 2617 [5].

4.1.2. CONTENT_LENGTH

The CONTENT_LENGTH variable contains the size of the message-body attached to the request, if any, in decimal number of octets. If no data is attached, then NULL (or unset).

CONTENT_LENGTH = “” | 1*digit

The server MUST set this meta-variable if and only if the request is accompanied by a message-body entity. The CONTENT_LENGTH value must reflect the length of the message-body after the server has removed any transfer-codings or content-codings.

4.1.3. CONTENT_TYPE

If the request includes a message-body, the CONTENT_TYPE variable is set to the Internet Media Type [6] of the message-body.

CONTENT_TYPE	=	“” | media-type
media-type	=	type “/” subtype *(“;” parameter)
type	=	token
subtype	=	token
parameter	=	attribute “=” value
attribute	=	token
value	=	token | quoted-string

The type, subtype and parameter attribute names are not case-sensitive. Parameter values may be case sensitive. Media types and their use in HTTP are described section 3.7 of the HTTP/1.1 specification [4].

There is no default value for this variable. If and only if it is unset, then the script MAY attempt to determine the media type from the data received. If the type remains unknown, then the script MAY choose to assume a type of application/octet-stream or it may reject the request with an error (as described in section 6.3.3).

Each media-type defines a set of optional and mandatory parameters. This may include a charset parameter with a case-insensitive value defining the coded character set for the message-body. If the charset parameter is omitted, then the default value should be derived according to whichever of the following rules is the first to apply:

1. There MAY be a system-defined default charset for some media-types.
2. The default for media-types of type “text” is ISO-8859-1 [4].
3. Any default defined in the media-type specification.
4. The default is US-ASCII.

The server MUST set this meta-variable if an HTTP Content-Type field is present in the client request header. If the server receives a request with an attached entity but no Content-Type header field, it MAY attempt to determine the correct content type, otherwise it should omit this meta-variable.

4.1.4. GATEWAY_INTERFACE

The GATEWAY_INTERFACE variable MUST be set to the dialect of CGI being used by the server to communicate with the script. Syntax:

GATEWAY_INTERFACE = “CGI” “/” 1*digit “.” 1*digit

Note that the major and minor numbers are treated as separate integers and hence each may be incremented higher than a single digit. Thus CGI/2.4 is a lower version than CGI/2.13 which in turn is lower than CGI/12.3. Leading zeros MUST be ignored by the script and MUST NOT be generated by the server.

This document defines the 1.1 version of the CGI interface.

4.1.5. PATH_INFO

The PATH_INFO variable specifies a path to be interpreted by the CGI script. It identifies the resource or sub-resource to be returned by the CGI script, and is derived from the portion of the URI path hierarchy following the part that identifies the script itself. Unlike a URI path, the PATH_INFO is not URL-encoded, and cannot contain path-segment parameters. A PATH_INFO of “/” represents a single void path segment.

PATH_INFO	=	“” | (“/” path)
path	=	lsegment *(“/” lsegment)
lsegment	=	*lchar
lchar	=	<any TEXT or CTL except “/”>

The value is considered case-sensitive and the server MUST preserve the case of the path as presented in the request URI. The server MAY impose restrictions and limitations on what values it permits for PATH_INFO, and MAY reject the request with an error if it encounters any values considered objectionable. That MAY include any requests that would result in an encoded “/” being decoded into PATH_INFO, as this might represent a loss of information to the script. Similarly, treatment of non US-ASCII characters in the path is system-defined.

URL-encoded, the PATH_INFO string forms the extra-path component of the Script-URI (see section 3.3) which follows the SCRIPT_NAME part of that path.

4.1.6. PATH_TRANSLATED

The PATH_TRANSLATED variable is derived by taking the PATH_INFO value, parsing it as a local URI in its own right, and performing any virtual-to-physical translation appropriate to map it onto the server's document repository structure. The set of characters permitted in the result is system-defined.

PATH_TRANSLATED = *<any character>

This is the file location that would be accessed by a request for

<scheme> "://" <server-name> ":" <server-port> <extra-path>

where <scheme> is the scheme for the original client request and <extra-path> is a URL-encoded version of PATH_INFO, with ";", "=" and "?" reserved. For example, a request such as the following:

http://somehost.com/cgi-bin/somescript/this%2eis%2epath%3binfo

would result in a PATH_INFO value of

/this.is.the.path;info

An internal URI is constructed from the scheme, server location and the URL-encoded PATH_INFO:

http://somehost.com/this.is.the.path%3binfo

This would then be translated to a location in the server's document repository, perhaps a filesystem path something like this:

/usr/local/www/htdocs/this.is.the.path;info

The value of PATH_TRANSLATED is the result of the translation.

The value is derived in this way irrespective of whether it maps to a valid repository location. The server MUST preserve the case of the extra-path segment unless the underlying repository supports case-insensitive names. If the repository is only case-aware, case-preserving, or case-blind with regard to document names, the server is not required to preserve the case of the original segment through the translation.

The translation algorithm the server uses to derive PATH_TRANSLATED is implementation-defined; CGI scripts which use this variable may suffer limited portability.

The server SHOULD set this meta-variable if the request URI includes a path-info component. If PATH_INFO is NULL, then the PATH_TRANSLATED variable MUST be set to NULL (or unset).

4.1.7. QUERY_STRING

The QUERY_STRING variable contains a URL-encoded search or parameter string; it provides information to the CGI script to affect or refine the document to be returned by the script.

The URL syntax for a search string is described in section 3 of RFC 2396 [2]. The QUERY_STRING value is case-sensitive.

```
QUERY_STRING    =    query-string
query-string    =    *uric
uric            =    reserved | unreserved | escaped
```

When parsing and decoding the query string, the details of the parsing, reserved characters and support for non US-ASCII characters depends on the context. For example, form submission from an HTML document [18] uses application/x-www-form-urlencoded encoding, in which the characters “+”, “&” and “=” are reserved, and the ISO 8859-1 encoding may be used for non US-ASCII characters.

The QUERY_STRING value provides the query-string part of the Script-URI. (See Section 8.1, "3.3. The Script-URI," on page 58).

The server MUST set this variable; if the Script-URI does not include a query component, the QUERY_STRING MUST be defined as an empty string (“”).

4.1.8. REMOTE_ADDR

The REMOTE_ADDR variable MUST be set to the network address of the client sending the request to the server.

```
REMOTE_ADDR     =    hostnumber
hostnumber      =    ipv4-address | ipv6-address
ipv4-address    =    1*3digit “.” 1*3digit “.” 1*3digit “.” 1*3digit
ipv6-address    =    hexpart [ “:” ipv4-address ]
hexpart         =    hexseq | ( [ hexseq ] “::” [ hexseq ] )
hexseq          =    1*4hex *( “:” 1*4hex )
```

The format of an IPv6 address is described in RFC 3513 [15].

4.1.9. REMOTE_HOST

The REMOTE_HOST variable contains the fully qualified domain name of the client sending the request to the server, if available, otherwise NULL. Fully qualified domain names take the form as described in section 3.5 of RFC 1034 [17] and section 2.1 of RFC 1123 [12]. Domain names are not case sensitive.

```
REMOTE_HOST     =    “” | hostname | hostnumber
hostname        =    *( domainlabel “.” ) toplabel [ “.” ]
domainlabel     =    alphanum [ *alphahypdigit alphanum ]
toplabel        =    alpha [ *alphahypdigit alphanum ]
alphahypdigit   =    alphanum | “-”
```

The server SHOULD set this variable. If the hostname is not available for performance reasons or otherwise, the server MAY substitute the REMOTE_ADDR value.

4.1.10. REMOTE_IDENT

The REMOTE_IDENT variable MAY be used to provide identity information reported about the connection by an RFC 1413 [20] request to the remote agent, if available. The server may choose not to support this feature, or not to request the data for efficiency reasons, or not to return available identity data.

REMOTE_IDENT = *TEXT

The data returned may be used for authentication purposes, but the level of trust reposed in it should be minimal.

4.1.11. REMOTE_USER

The REMOTE_USER variable provides a user identification string supplied by client as part of user authentication.

REMOTE_USER = *TEXT

If the client request required HTTP Authentication [5] (e.g., the AUTH_TYPE meta-variable is set to “Basic” or “Digest”), then the value of the REMOTE_USER meta-variable MUST be set to the user-ID supplied.

4.1.12. REQUEST_METHOD

The REQUEST_METHOD meta-variable MUST be set to the method which should be used by the script to process the request, as described in section 4.3.

REQUEST_METHOD	=	method
method	=	“GET” | “POST” | “HEAD” | extension-method
extension-method	=	“PUT” | “DELETE” | token

The method is case sensitive. The HTTP methods are described in section 5.1.1 of the HTTP/1.0 specification [1] and section 5.1.1 of the HTTP/1.1 specification [4].

4.1.13. SCRIPT_NAME

The SCRIPT_NAME variable MUST be set to a URI path (not URL-encoded) which could identify the CGI script (rather than the script’s output). The syntax is the same as for PATH_INFO (section 4.1.5)

SCRIPT_NAME = “” | (“/” path)

The leading “/” is not part of the path. It is optional if the path is NULL; however, the variable MUST still be set in that case.

The SCRIPT_NAME string forms some leading part of the path component of the Script-URI derived in some implementation-defined manner. No PATH_INFO segment (see section 4.1.5) is included in the SCRIPT_NAME value.

4.1.14. SERVER_NAME

The SERVER_NAME variable MUST be set to the name of the server host to which the client request is directed. It is a case-insensitive hostname or network address. It forms the host part of the Script-URI.

```
SERVER_NAME     =   server-name
server-name     =   hostname | ipv4-address | ( "[" ipv6-address "]" )
```

A deployed server can have more than one possible value for this variable, where several HTTP virtual hosts share the same IP address. In that case, the server would use the contents of the request's Host header field to select the correct virtual host.

4.1.15. SERVER_PORT

The SERVER_PORT variable MUST be set to the TCP/IP port number on which this request is received from the client. This value is used in the port part of the Script-URI.

```
SERVER_PORT     =   server-port
server-port     =   1*digit
```

Note that this variable MUST be set, even if the port is the default port for the scheme and could otherwise be omitted from a URI.

4.1.16. SERVER_PROTOCOL

The SERVER_PROTOCOL variable MUST be set to the name and version of the application protocol used for this CGI request. This MAY differ from the protocol version used by the server in its communication with the client.

```
SERVER_PROTOCOL   =   HTTP-Version | "INCLUDED" | extension-version
HTTP-Version      =   "HTTP" "/" 1*digit "." 1*digit
extension-version =   protocol [ "/" 1*digit "." 1*digit ]
protocol          =   token
```

Here, 'protocol' defines the syntax of some of the information passing between the server and the script (the 'protocol-specific' features). It is not case sensitive and is usually presented in upper case. The protocol is not the same as the scheme part of the script URI, which defines the overall access mechanism used by the client to communicate with the server. For example, a request that reaches the script with a protocol of "HTTP" may have used an "https" scheme.

A well-known value for SERVER_PROTOCOL which the server MAY use is "INCLUDED", which signals that the current document is being included as part of a composite document, rather than being the direct target of the client request. The script should treat this as an HTTP/1.0 request.

4.1.17. SERVER_SOFTWARE

The SERVER_SOFTWARE meta-variable MUST be set to the name and version of the information server software making the CGI request (and running the gateway). It SHOULD be the same as the server description reported to the client, if any.

SERVER_SOFTWARE	=	1*(product | comment)
product	=	token ["/" product-version]
product-version	=	token
comment	=	"(" *(ctext | comment) ")"
ctext	=	<any TEXT excluding "(" and ")">

4.1.18. Protocol-Specific Meta-Variables

The server SHOULD set meta-variables specific to the protocol and scheme for the request. Interpretation of protocol-specific variables depends on the protocol version in SERVER_PROTOCOL. The server MAY set a meta-variable with the name of the scheme to a non-NULL value if the scheme is not the same as the protocol. The presence of such a variable indicates to a script which scheme is used by the request.

Meta-variables with names beginning with "HTTP_" contain values read from the client request header fields, if the protocol used is HTTP. The HTTP header field name is converted to upper case, has all occurrences of "-" replaced with "_" and has "HTTP_" prepended to give the meta-variable name. The header data can be presented as sent by the client, or can be rewritten in ways which do not change its semantics. If multiple header fields with the same field-name are received then the server MUST rewrite them as a single value having the same semantics. Similarly, a header field that spans multiple lines MUST be merged onto a single line. The server MUST, if necessary, change the representation of the data (for example, the character set) to be appropriate for a CGI meta-variable.

The server is not required to create meta-variables for all the header fields that it receives. In particular, it SHOULD remove any header fields carrying authentication information, such as 'Authorization'; or that are available to the script in other variables, such as 'Content-Length' and 'Content-Type'. The server MAY remove header fields that relate solely to client-side communication issues, such as 'Connection'.

4.2. Request Message-Body

Request data is accessed by the script in a system-defined method; unless defined otherwise, this will be by reading the 'standard input' file descriptor or file handle.

Request-Data	=	[request-body] [extension-data]
request-body	=	<CONTENT_LENGTH>OCTET
extension-data	=	*OCTET

A request-body is supplied with the request if the CONTENT_LENGTH is not NULL. The server MUST make at least that many bytes available for the script to read. The server MAY signal an end-of-file condition after CONTENT_LENGTH bytes have been read or it MAY supply extension data. Therefore, the script MUST NOT attempt to read

more than CONTENT_LENGTH bytes, even if more data is available. However, it is not obliged to read any of the data.

For non-parsed header (NPH) scripts (section 5), the server SHOULD attempt to ensure that the data supplied to the script is precisely as supplied by the client and is unaltered by the server.

As transfer-codings are not supported on the request-body, the server MUST remove any such codings from the message-body, and recalculate the CONTENT_LENGTH. If this is not possible (for example, because of large buffering requirements), the server SHOULD reject the client request. It MAY also remove content-codings from the message-body.

4.3. Request Methods

The Request Method, as supplied in the REQUEST_METHOD meta-variable, identifies the processing method to be applied by the script in producing a response. The script author can choose to implement the methods most appropriate for the particular application. If the script receives a request with a method it does not support it SHOULD reject it with an error (see section 6.3.3).

4.3.1. GET

The GET method indicates that the script should produce a document based on the meta-variable values. By convention, the GET method is ‘safe’ and ‘idempotent’ and SHOULD NOT have the significance of taking an action other than producing a document.

The meaning of the GET method may be modified and refined by protocol-specific meta-variables.

4.3.2. POST

The POST method is used to request the script perform processing and produce a document based on the data in the request message-body, inaddition to meta-variable values. A common use is form submission in HTML [18], intended to initiate processing by the script that has a permanent affect, such a change in a database.

The script MUST check the value of the CONTENT_LENGTH variable before reading the attached message-body, and SHOULD check the CONTENT_TYPE value before processing it.

4.3.3. HEAD

The HEAD method requests the script to do sufficient processing to return the response header fields, without providing a response message-body. The script MUST NOT provide a response message-body for a HEAD request. If it does, then the server MUST discard the message-body when reading the response from the script.

4.3.4. Protocol-Specific Methods

The script MAY implement any protocol-specific method, such as HTTP/1.1 PUT and DELETE; it SHOULD check the value of SERVER_PROTOCOL when doing so.

The server MAY decide that some methods are not appropriate or permitted for a script, and may handle the methods itself or return an error to the client.

4.4. The Script Command Line

Some systems support a method for supplying an array of strings to the CGI script. This is only used in the case of an 'indexed' HTTP query, which is identified by a 'GET' or 'HEAD' request with a URI query string that does not contain any unencoded "=" characters. For such a request, the server SHOULD treat the query-string as a search-string and parse it into words, using the rules

search-string	=	search-word *("+" search-word)
search-word	=	1*schar
schar	=	unreserved | escaped | xreserved
xreserved	=	";" | "/" | "?" | ":" | "@" | "&" | "=" | "," | "$"

After parsing, each search-word is URL-decoded, optionally encoded in a system-defined manner and then added to the command line argument list.

If the server cannot create any part of the argument list, then the server MUST NOT generate any command line information. For example, the number of arguments may be greater than operating system or server limits, or one of the words may not be representable as an argument.

The script SHOULD check to see if the QUERY_STRING value contains an unencoded "=" character, and SHOULD NOT use the command line arguments if it does.

5. NPH Scripts

5.1. Identification

The server MAY support NPH (Non-Parsed Header) scripts; these are scripts to which the server passes all responsibility for response processing.

This specification provides no mechanism for an NPH script to be identified on the basis of its output data alone. By convention, therefore, any particular script can only ever provide output of one type (NPH or CGI) and hence the script itself is described as an 'NPH script'. A server with NPH support MUST provide an implementation-defined mechanism for identifying NPH scripts, perhaps based on the name or location of the script.

5.2. NPH Response

There MUST be a system-defined method for the script to send data back to the server or client; a script MUST always return some data. Unless defined otherwise, this will be the same as for conventional CGI scripts.

Currently, NPH scripts are only defined for HTTP client requests. An (HTTP) NPH script MUST return a complete HTTP response message, currently described in section 6 of the HTTP specifications [1], [4]. The script MUST use the SERVER_PROTOCOL variable to determine the appropriate format for a response. It MUST also take account of any generic or protocol-specific meta-variables in the request as might be mandated by the particular protocol specification.

The server MUST ensure that the script output is sent to the client unmodified. Note that this requires the script to use the correct character set (US-ASCII [9] and ISO 8859-1 [10] for HTTP) in the header fields. The server SHOULD attempt to ensure that the script output is sent directly to the client, with minimal internal and no transport-visible buffering.

Unless the implementation defines otherwise, the script MUST NOT indicate in its response that the client can send further requests over the same connection.

6. CGI Response

6.1. Response Handling

A script MUST always provide a non-empty response, and so there is a system-defined method for it to send this data back to the server. Unless defined otherwise, this will be via the ‘standard output’ file descriptor.

The script MUST check the REQUEST_METHOD variable when processing the request and preparing its response.

The server MAY implement a timeout period within which data must be received from the script. If a server implementation defines such a timeout and receives no data from a script within the timeout period, the server MAY terminate the script process.

6.2. Response Types

The response comprises a message-header and a message-body, separated by a blank line. The message-header contains one or more header fields. The body may be NULL.

generic-response = 1*header-field NL [response-body]

The script MUST return one of either a document response, a local redirect response or a client redirect (with optional document) response. In the response definitions below, the order of header fields in a response is not significant (despite appearing so in the BNF). The header fields are defined in section 6.3.

CGI-Response = document-response | local-redir-response |
client-redir-response | client-redirdoc-response

6.2.1. Document Response

The CGI script can return a document to the user in a document response, with an optional error code indicating the success status of the response.

document-response = Content-Type [Status] *other-field NL
response-body

The script MUST return a Content-Type header field. A Status header field is optional, and status 200 ‘OK’ is assumed if it is omitted. The server MUST make any appropriate modifications to the script’s output to ensure that the response to the client complies with the response protocol version.

6.2.2. Local Redirect Response

The CGI script can return a URI path and query-string (‘local-pathquery’) for a local resource in a Location header field. This indicates to the server that it should reprocess the request using the path specified.

local-redir-response = local-Location NL

The script MUST NOT return any other header fields or a message-body, and the server MUST generate the response that it would have produced in response to a request containing the URL

scheme “://” server-name “:” server-port local-pathquery

6.2.3. Client Redirect Response

The CGI script can return an absolute URI path in a Location header field, to indicate to the client that it should reprocess the request using the URI specified.

client-redir-response = client-Location *extension-field NL

The script MUST not provide any other header fields, except for server-defined CGI extension fields. For an HTTP client request, the server MUST generate a 302 ‘Found’ HTTP response message.

6.2.4. Client Redirect Response with Document

The CGI script can return an absolute URI path in a Location header field together with an attached document, to indicate to the client that it should reprocess the request using the URI specified.

client-redirdoc-response = client-Location Status Content-Type
*other-field NL response-body

The Status header field MUST be supplied and MUST contain a status value of 302 'Found', or it MAY contain an extension-code, that is, another valid status code that means client redirection. The server MUST make any appropriate modifications to the script's output to ensure that the response to the client complies with the response protocol version.

6.3. Response Header Fields

The response header fields are either CGI or extension header fields to be interpreted by the server, or protocol-specific header fields to be included in the response returned to the client. At least one CGI field MUST be supplied; each CGI field MUST NOT appear more than once in the response. The response header fields have the syntax:

header-field	=	CGI-field | other-field
CGI-field	=	Content-Type | Location | Status
other-field	=	protocol-field | extension-field
protocol-field	=	generic-field
extension-field	=	generic-field
generic-field	=	field-name ":" [field-value] NL
field-name	=	token
field-value	=	*(field-content | LWSP)
field-content	=	*(token | separator | quoted-string)

The field-name is not case sensitive. A NULL field value is equivalent to a field not being sent. Note that each header field in a CGI-Response MUST be specified on a single line; CGI/1.1 does not support continuation lines. Whitespace is permitted between the ":" and the field-value (but not between the field-name and the ":"), and also between tokens in the field-value.

6.3.1. Content-Type

The Content-Type response field sets the Internet Media Type [6] of the entity body.

Content-Type	=	"Content-Type:" media-type NL

If an entity body is returned, the script MUST supply a Content-Type field in the response. If it fails to do so, the server SHOULD NOT attempt to determine the correct content type. The value SHOULD be sent unmodified to the client, except for any charset parameter changes.

Unless it is otherwise system-defined, the default charset assumed by the client for text media-types is ISO-8859-1 if the protocol is HTTP and US-ASCII otherwise. Hence the script SHOULD include a charset parameter. See section 3.4.1 of the HTTP/1.1 specification [4] for a discussion of this issue.

6.3.2. Location

The Location header field is used to specify to the server that the script is returning a reference to a document rather than an actual document (see sections 6.2.3 and 6.2.4). It is either an absolute URI (optionally with a fragment identifier), indicating that the client

is to fetch the referenced document, or a local URI path (optionally with a query string), indicating that the server is to fetch the referenced document and return it to the client as the response.

Location	=	local-Location | client-Location
client-Location	=	“Location:” fragment-URI NL
local-Location	=	“Location:” local-pathquery NL
fragment-URI	=	absoluteURI[“#” fragment]
fragment	=	*uric
local-pathquery	=	abs-path [“?” query-string]
abs-path	=	“/” path-segments
path-segments	=	segment *(“/” segment)
segment	=	*pchar
pchar	=	unreserved | escaped | extra
extra	=	“:” | “@” | “&” | “=” | “+” | “$” | “,”

The syntax of an absoluteURI is incorporated into this document from that specified in RFC 2396 [2] and RFC 2732 [7]. A valid absoluteURI always starts with the name of scheme followed by “:”; scheme names start with a letter and continue with alphanumerics, “+”, “-” or “.”. The local URI path and query must be an absolute path, and not a relative path or NULL, and hence must start with a “/”.

Note that any message-body attached to the request (such as for a POST request) may not be available to the resource that is the target of the redirect.

6.3.3. Status

The Status header field contains a 3-digit integer result code that indicates the level of success of the script’s attempt to handle the request.

Status	=	“Status:” status-code SP reason-phrase NL
status-code	=	“200” | “302” | “400” | “501” | extension-code
extension-code	=	3digit
reason-phrase	=	*TEXT

Status code 200 ‘OK’ indicates success, and is the default value assumed for a document response. Status code 302 ‘Found’ is used with a Location header field and response message-body. Status code 400 ‘Bad Request’ may be used for an unknown request format, such as a missing CONTENT_TYPE. Status code 501 ‘Not Implemented’ may be returned by a script if it receives an unsupported REQUEST_METHOD.

Other valid status codes are listed in section 6.1.1 of the HTTP specifications [1], [4], and also the IANA HTTP Status Code Registry [8] and MAY be used in addition to or instead of the ones listed above. The script SHOULD check the value of SERVER_PROTOCOL before using HTTP/1.1 status codes. The script MAY reject with error 405 ‘Method Not Allowed’ HTTP/1.1 requests made using a method it does not support.

Note that returning an error status code does not have to mean an error condition with the script itself. For example, a script that is invoked as an error handler by the server should return the code appropriate to the server’s error condition.

The reason-phrase is a textual description of the error to be returned to the client for human consumption.

6.3.4. Protocol-Specific Header Fields

The script MAY return any other header fields that relate to the response message defined by the specification for the SERVER_PROTOCOL (HTTP/1.0 [1] or HTTP/1.1 [4]). The server MUST translate the header data from the CGI header syntax to the HTTP header syntax if these differ. For example, the character sequence for newline (such as UNIX's US-ASCII LF) used by CGI scripts may not be the same as that used by HTTP (US-ASCII CR followed by LF).

The script MUST NOT return any header fields that relate to client-side communication issues and could affect the server's ability to send the response to the client. The server MAY remove any such header fields returned by the client. It SHOULD resolve any conflicts between header fields returned by the script and header fields that it would otherwise send itself.

6.3.5. Extension Header Fields

There may be additional implementation-defined CGI header fields, whose field names SHOULD begin with "X-CGI-". The server MAY ignore (and delete) any unrecognised header fields with names beginning "X-CGI-" that are received from the script.

6.4. Response Message-Body

The response message-body is an attached document to be returned to the client by the server. The server MUST read all the data provided by the script, until the script signals the end of the message-body by way of an end-of-file condition. The message-body SHOULD be sent unmodified to the client, except for HEAD requests or any required transfer-codings, content-codings or charset conversions.

response-body = *OCTET

7. System Specifications

7.1. AmigaDOS

Meta-Variables

Meta-variables are passed to the script in identically named environment variables. These are accessed by the DOS library routine *GetVar()*. The flags argument SHOULD be 0. Case is ignored, but upper case is recommended for compatibility with case-sensitive systems.

The current working directory

The current working directory for the script is set to the directory containing the script.

Character set

The US-ASCII character set [9] is used for the definition of meta-variables, header fields and values; the newline (NL) sequence is LF; servers SHOULD also accept CR LF as a newline.

7.2. UNIX

For UNIX compatible operating systems, the following are defined:

Meta-Variables

Meta-variables are passed to the script in identically named environment variables. These are accessed by the C library routine *getenv()* or variable *environ*.

The command line

This is accessed using the argc and argv arguments to *main()*. The words have any characters which are ‘active’ in the Bourne shell escaped with a backslash.

The current working directory

The current working directory for the script SHOULD be set to the directory containing the script.

Character set

The US-ASCII character set [9], excluding NUL, is used for the definition of meta-variables, header fields and CHAR values; TEXT values use ISO-8859-1. The PATH_TRANSLATED value can contain any 8-bit byte except NUL. The newline (NL) sequence is LF; servers should also accept CR LF as a newline.

7.3. EBCDIC/POSIX

For POSIX compatible operating systems using the EBCDIC character set, the following are defined:

Meta-Variables

Meta-variables are passed to the script in identically named environment variables. These are accessed by the C library routine *getenv()*.

The command line

This is accessed using the argc and argv arguments to *main()*. The words have any characters which are ‘active’ in the Bourne shell escaped with a backslash.

The current working directory

The current working directory for the script SHOULD be set to the directory containing the script.

Character set

The IBM1047 character set [21], excluding NUL, is used for the definition of meta-variables, header fields, values, TEXT strings and the PATH_TRANSLATED value. The newline (NL) sequence is LF; servers should also accept CR LF as a newline.

media-type charset default

The default charset value for text (and other implementation-defined) media types is IBM1047.

8. Implementation

8.1. Recommendations for Servers

Although the server and the CGI script need not be consistent in their handling of URL paths (client URLs and the PATH_INFO data, respectively), server authors may wish to impose consistency. So the server implementation should specify its behaviour for the following cases:

1. define any restrictions on allowed path segments, in particular whether non-terminal NULL segments are permitted;
2. define the behaviour for "." or ".." path segments; i.e., whether they are prohibited, treated as ordinary path segments or interpreted in accordance with the relative URL specification [2];
3. define any limits of the implementation, including limits on path or search string lengths, and limits on the volume of header fields the server will parse.

8.2. Recommendations for Scripts

If the script does not intend processing the PATH_INFO data, then it should reject the request with 404 Not Found if PATH_INFO is not NULL.

If the output of a form is being processed, check that CONTENT_TYPE is "application/x-www-form-urlencoded" [18] or "multipart/form-data" [16]. If CONTENT_TYPE is blank, the script can reject the request with a 415 'Unsupported Media Type' error, where supported by the protocol.

When parsing PATH_INFO, PATH_TRANSLATED or SCRIPT_NAME the script should be careful of void path segments ("//") and special path segments ("." and ".."). They should either be removed from the path before use in OS system calls, or the request should be rejected with 404 'Not Found'.

When returning header fields, the script should try to send the CGI header fields as soon as possible, and should send them before any HTTP header fields. This may help reduce the server's memory requirements.

Script authors should be aware that the REMOTE_ADDR and REMOTE_HOST meta-variables (see sections 4.1.8 and 4.1.9) may not identify the ultimate source of the request. They identify the client for the immediate request to the server; that client may be a proxy, gateway, or other intermediary acting on behalf of the actual source client.

9. Security Considerations

9.1. Safe Methods

As discussed in the security considerations of the HTTP specifications [1], [4], the convention has been established that the GET and HEAD methods should be ‘safe’ and ‘idempotent’ (repeated requests have the same effect as a single request). See section 9.1 of RFC 2616 [4] for a full discussion.

9.2. Header Fields Containing Sensitive Information

Some HTTP header fields may carry sensitive information which the server should not pass on to the script unless explicitly configured to do so. For example, if the server protects the script by using the Basic authentication scheme, then the client will send an Authorization header field containing a username and password. The server validates this information and so it should not pass on the password via the HTTP_AUTHORIZATION meta-variable without careful consideration. This also applies to the Proxy-Authorization header field and the corresponding HTTP_PROXY_AUTHORIZATION meta-variable.

9.3. Data Privacy

Confidential data in a request should be placed in a message-body as part of a POST request, and not placed in the URI or message headers. On some systems, the environment used to pass meta-variables to a script may be visible to other scripts or users. In addition, many existing servers, proxies and clients will permanently record the URI where it might be visible to third parties.

9.4. Information Security Model

For a client connection using TLS, the security model applies between the client and the server, and not between the client and the script. It is the server’s responsibility to handle the TLS session, and thus it is the server which is authenticated to the client, not the CGI script.

This specification provides no mechanism for the script to authenticate the server which invoked it. There is no enforced integrity on the CGI request and response messages.

9.5. Script Interference with the Server

The most common implementation of CGI invokes the script as a child process using the same user and group as the server process. It should therefore be ensured that the script cannot interfere with the server process, its configuration, documents or log files.

If the script is executed by calling a function linked in to the server software (either at compile-time or run-time) then precautions should be taken to protect the core memory of the server, or to ensure that untrusted code cannot be executed.

9.6. Data Length and Buffering Considerations

This specification places no limits on the length of the message-body presented to the script. The script should not assume that statically allocated buffers of any size are sufficient to contain the entire submission at one time. Use of a fixed length buffer without careful overflow checking may result in an attacker exploiting 'stack-smashing' or 'stack-overflow' vulnerabilities of the operating system. The script may spool large submissions to disk or other buffering media, but a rapid succession of large submissions may result in denial of service conditions. If the CONTENT_LENGTH of a message-body is larger than resource considerations allow, scripts should respond with an error status appropriate for the protocol version; potentially applicable status codes include 503 'Service Unavailable' (HTTP/1.0 and HTTP/1.1), 413 'Request Entity Too Large' (HTTP/1.1), and 414 'Request-URI Too Large' (HTTP/1.1).

Similar considerations apply to the server's handling of the CGI response from the script. There is no limit on the length of the header or message-body returned by the script; the server should not assume that statically allocated buffers of any size are sufficient to contain the entire response.

9.7. Stateless Processing

The stateless nature of the Web makes each script execution and resource retrieval independent of all others even when multiple requests constitute a single conceptual Web transaction. Because of this, a script should not make any assumptions about the context of the user-agent submitting a request. In particular, scripts should examine data obtained from the client and verify that they are valid, both in form and content, before allowing them to be used for sensitive purposes such as input to other applications, commands, or operating system services. These uses include (but are not limited to) system call arguments, database writes, dynamically evaluated source code, and input to billing or other secure processes. It is important that applications be protected from invalid input regardless of whether the invalidity is the result of user error, logic error, or malicious action.

Authors of scripts involved in multi-request transactions should be particularly cautious about validating the state information; undesirable effects may result from the substitution of dangerous values for portions of the submission which might otherwise be presumed safe. Subversion of this type occurs when alterations are made to data from a prior stage of the transaction that were not meant to be controlled by the client (e.g., hidden HTML form elements, cookies, embedded URLs, etc.).

9.8. Relative Paths

The server should be careful of ".." path segments in the request URI. These should be removed or resolved in the request URI before it is split into the script-path and extra-path. Alternatively, when the extra-path is used to find the PATH_TRANSLATED, care should be taken to avoid the path resolution from providing translated paths outside an expected path hierarchy.

9.9. Non-parsed Header Output

If a script returns a non-parsed header output, to be interpreted by the client in its native protocol, then the script must address all security considerations relating to that protocol.

10. Acknowledgements

This work is based on the original CGI interface that arose out of discussions on the 'www-talk' mailing list. In particular, Rob McCool, John Franks, Ari Luotonen, George Phillips and Tony Sanders deserve special recognition for their efforts in defining and implementing the early versions of this interface.

This document has also greatly benefited from the comments and suggestions made Chris Adie, Dave Kristol and Mike Meyer; also David Morris, Jeremy Madea, Patrick McManus, Adam Donahue, Ross Patterson and Harald Alvestrand.

11. References

11.1 Normative References

[1] Berners-Lee, T., Fielding, R. and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[2] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI) : Generic Syntax", RFC 2396, August 1998.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirements Levels", BCP 14, RFC 2119, March 1997.

[4] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[5] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[6] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[7] Hinden, R., Carpenter, B., and L. Masinter, “Format for Literal IPv6 Addresses in URL’s”, RFC 2732, December 1999.

[8] “HTTP Status Code Registry”, http://www.iana.org/assignments/http-status-codes, IANA.

[9] “Information Systems -- Coded Character Sets -- 7-bit American Standard Code for Information Interchange (7-Bit ASCII)”, ANSI INCITS.4-1986 (R2002).

[10] “Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1”, ISO/IEC 8859-1:1998.

11.2. Informative References

[11] Berners-Lee, T., “Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web”, RFC 1630, June 1994.

[12] Braden, R., Ed., “Requirements for Internet Hosts -- Application and Support”, STD 3, RFC 1123, October 1989.

[13] Crocker, D., “Standard for the Format of ARPA Internet Text Messages”, STD 11, RFC 822, August 1982.

[14] Dierks, T. and C. Allen, “The TLS Protocol Version 1.0”, RFC 2246, January 1999.

[15] Hinden R. and S. Deering, “Internet Protocol Version 6 (IPv6) Addressing Architecture”, RFC 3513, April 2003.

[16] Masinter, L., “Returning Values from Forms: multipart/form-data”, RFC 2388, August 1998.

[17] Mockapetris, P., “Domain Names - Concepts and Facilities”, STD 13, RFC 1034, November 1987.

[18] Raggett, D., Le Hors, A., and I. Jacobs, Eds., “HTML 4.01 Specification”, W3C Recommendation December 1999, http://www.w3.org/TR/html401/.

[19] Rescola, E. “HTTP Over TLS”, RFC 2818, May 2000.

[20] St. Johns, M., “Identification Protocol”, RFC 1413, February 1993.

[21] IBM National Language Support Reference Manual Volume 2, SE09-8002-01, March 1990.

[22] “The Common Gateway Interface”, http://hoohoo.ncsa.uiuc.edu/cgi/, NCSA, University of Illinois.

12. Authors' Addresses

David Robinson The Apache Software Foundation
EMail: drtr@apache.org
Ken A. L. Coar The Apache Software Foundation
EMail: coar@apache.org

13. Full Copyright Statement

Html markup produced by *rfcmarkup* 1.66, available from *http://tools.ietf.org/tools/rfcmarkup/*

Appendix C: The application/json Media Type for JavaScript Object Notation (JSON)

Included, courtesy of:
Network Working Group D. *Crockford*
Request for Comments: 4627 *JSON.org*
Category: Informational *July 2006*

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data.

1. Introduction

JavaScript Object Notation (JSON) is a text format for the serialization of structured data. It is derived from the object literals of JavaScript, as defined in the ECMAScript Programming Language Standard, Third Edition [ECMA].

JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).

A string is a sequence of zero or more Unicode characters [UNICODE].

An object is an unordered collection of zero or more name/value pairs, where a name is a string and a value is a string, number, boolean, null, object, or array.

An array is an ordered sequence of zero or more values.

The terms “object” and “array” come from the conventions of JavaScript.

JSON’s design goals were for it to be minimal, portable, textual, and a subset of JavaScript.

1.1. Conventions Used in This Document

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

The grammatical rules in this document are to be interpreted as described in [RFC4234].

2. JSON Grammar

A JSON text is a sequence of tokens. The set of tokens includes six structural characters, strings, numbers, and three literal names.

A JSON text is a serialized object or array.

```
JSON-text = object / array
```

These are the six structural characters:

```
begin-array     = ws %x5B ws ;  [  left square bracket
begin-object    = ws %x7B ws ;  {  left curly bracket
end-array       = ws %x5D ws ;  ]  right square bracket
end-object      = ws %x7D ws ;  }  right curly bracket
name-separator  = ws %x3A ws ;  :  colon
value-separator = ws %x2C ws ;  ,  comma
```

Insignificant whitespace is allowed before or after any of the six structural characters.

```
ws = *(
        %x20        /  ;  Space
        %x09        /  ;  Horizontal tab
        %x0A        /  ;  Line feed or New line
        %x0D           ;  Carriage return
      )
```

2.1. Values

A JSON value MUST be an object, array, number, or string, or one of the following three literal names:

```
false null true
```

The literal names MUST be lowercase. No other literal names are allowed.

```
value    =  false / null / true / object / array / number / string
false    =  %x66.61.6c.73.65     ;  false
null     =  %x6e.75.6c.6c        ;  null
true     =  %x74.72.75.65        ;  true
```

2.2. Objects

An object structure is represented as a pair of curly brackets surrounding zero or more name/value pairs (or members). A name is a string. A single colon comes after each name, separating the name from the value. A single comma separates a value from a following name. The names within an object SHOULD be unique.

```
object    =  begin-object [ member *( value-separator member ) ] end-object
member    =  string name-separator value
```

2.3. Arrays

An array structure is represented as square brackets surrounding zero or more values (or elements). Elements are separated by commas.

```
array     =  begin-array [ value *( value-separator value ) ] end-array
```

2.4. Numbers

The representation of numbers is similar to that used in most programming languages. A number contains an integer component that may be prefixed with an optional minus sign, which may be followed by a fraction part and/or an exponent part.

Octal and hex forms are not allowed. Leading zeros are not allowed.

A fraction part is a decimal point followed by one or more digits.

An exponent part begins with the letter E in upper or lowercase, which may be followed by a plus or minus sign. The E and optional sign are followed by one or more digits.

Numeric values that cannot be represented as sequences of digits (such as Infinity and NaN) are not permitted.

```
number         =  [ minus ] int [ frac ] [ exp ]
decimal-point  =  %x2E                 ;  .
digit1-9       =  %x31-39              ;  1-9
e              =  %x65 / %x45          ;  e E
exp            =  e [ minus / plus ] 1*DIGIT
```

```
frac            =   decimal-point 1*DIGIT
int             =   zero / ( digit1-9 *DIGIT )
minus           =   %x2D                ;           -
plus            =   %x2B                ;           +
zero            =   %x30                ;           0
```

2.5. Strings

The representation of strings is similar to conventions used in the C family of programming languages. A string begins and ends with quotation marks. All Unicode characters may be placed within the quotation marks except for the characters that must be escaped: quotation mark, reverse solidus, and the control characters (U+0000 through U+001F).

Any character may be escaped. If the character is in the Basic Multilingual Plane (U+0000 through U+FFFF), then it may be represented as a six-character sequence: a reverse solidus, followed by the lowercase letter *u*, followed by four hexadecimal digits that encode the character's code point. The hexadecimal letters *A* though *F* can be upper or lowercase. So, for example, a string containing only a single reverse solidus character may be represented as "*\u005C*".

Alternatively, there are two-character sequence escape representations of some popular characters. So, for example, a string containing only a single reverse solidus character may be represented more compactly as "\\".

To escape an extended character that is not in the Basic Multilingual Plane, the character is represented as a twelve-character sequence, encoding the UTF-16 surrogate pair. So, for example, a string containing only the *G clef* character (*U+1D11E*) may be represented as "*\uD834\uDD1E*".

```
string      =           quotation-mark *char quotation-mark

char        =           unescaped /
      escape (
      %x22 /            ;           " quotation mark    U+0022
      %x5C/             ; \         reverse solidus     U+005C
      %x2F /            ; /         solidus             U+002F
      %x62/             ; b         backspace           U+0008
      %x66/             ; f         form feed           U+000C
      %x6E /            ; n         line feed           U+000A
      %x72 /            ; r         carriage return     U+000D
      %x74 / ; t        tab         U+0009
      %x75 4HEXDIG ) ;              uXXXX U+XXXX

escape              =   %x5C ; \

quotation-mark      =   %x22 ; "

 unescaped          =   %x20-21 /       %x23-5B       / %x5D-10FFFF
```

3. Encoding

JSON text SHALL be encoded in Unicode. The default encoding is UTF-8.

Since the first two characters of a JSON text will always be ASCII characters [RFC0020], it is possible to determine whether an octet stream is UTF-8, UTF-16 (BE or LE), or UTF-32 (BE or LE) by looking at the pattern of nulls in the first four octets.

00	00	00	xx	UTF-32BE
00	xx	00	xx	UTF-16BE
xx	00	00	00	UTF-32LE
xx	00	xx	00	UTF-16LE
xx	xx	xx	xx	UTF-8

4. Parsers

A JSON parser transforms a JSON text into another representation. A JSON parser MUST accept all texts that conform to the JSON grammar. A JSON parser MAY accept non-JSON forms or extensions.

An implementation may set limits on the size of texts that it accepts. An implementation may set limits on the maximum depth of nesting. An implementation may set limits on the range of numbers. An implementation may set limits on the length and character contents of strings.

5. Generators

A JSON generator produces JSON text. The resulting text MUST strictly conform to the JSON grammar.

6. IANA Considerations

The MIME media type for JSON text is application/json.

Type name: application

Subtype name: json

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: 8bit if UTF-8; binary if UTF-16 or UTF-32

JSON may be represented using UTF-8, UTF-16, or UTF-32. When JSON is written in UTF-8, JSON is 8bit compatible. When JSON is written in UTF-16 or UTF-32, the binary content-transfer-encoding must be used.

Security considerations:

Generally there are security issues with scripting languages. JSON is a subset of JavaScript, but it is a safe subset that excludes assignment and invocation.

A JSON text can be safely passed into JavaScript's eval() function (which compiles and executes a string) if all the characters not enclosed in strings are in the set of characters that form JSON tokens. This can be quickly determined in JavaScript with two regular expressions and calls to the test and replace methods.

```
var my_JSON_object =
        !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test( text.replace(/"(\\.|[^"\\])*"/g, '')))
        && eval('(' + text + ')');
```

Interoperability considerations: n/a

Applications that use this media type:

JSON has been used to exchange data between applications written in all of these programming languages: ActionScript, C, C#, ColdFusion, Common Lisp, E, Erlang, Java, JavaScript, Lua, Objective CAML, Perl, PHP, Python, Rebol, Ruby, and Scheme.

Additional information:

Magic number(s): n/a

File extension(s): .json

Macintosh file type code(s): TEXT

Person & email address to contact for further information:

Douglas Crockford
douglas@crockford.com

Intended usage: COMMON

Restrictions on usage: none

Author:

Douglas Crockford
douglas@crockford.com

Change controller:

Douglas Crockford
douglas@crockford.com

7. Security Considerations

See Security Considerations in Section 6.

8. Examples

This is a JSON object:

```
{
        “Image”         :                {
        “Width”:        800,
        “Height”:       600,
        “Title”:        “View from 15th Floor”,
        “Thumbnail”:    {
                        “Url”:          “http://www.example.com/image/481989943”,
                        “Height”:       125,
                        “Width”:        “100”
        },
        “IDs”: [116, 943, 234, 38793]
        }
}
```

Its Image member is an object whose Thumbnail member is an object and whose IDs member is an array of numbers.

This is a JSON array containing two objects:

```
[
    {
            “precision” :   “zip”,
            “Latitude” :    37.7668,
            “Longitude” :   -122.3959,
            “Address” :     “”,
            “City” :        “SAN FRANCISCO”,
            “State” :       “CA”,
            “Zip” :         “94107”,
            “Country” :     “US”
    },
    {
            “precision” :   “zip”,
            “Latitude” :    37.371991,
            “Longitude” :   -122.026020,
            “Address” :     “”,
            “City” :        “SUNNYVALE”,
            “State” :       “CA”,
            “Zip” :         “94085”,
            “Country” :     “US”
    }

]
```

9. References

9.1. Normative References

[ECMA] European Computer Manufacturers Association, “ECMAScript Language Specification 3rd Edition”, December 1999, <*http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf*>.

[RFC0020] Cerf, V., “ASCII format for network interchange”, RFC 20, October 1969.

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997.

[RFC4234] Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF”, RFC 4234, October 2005.

[UNICODE] The Unicode Consortium, “The Unicode Standard Version 4.0”, 2003, <h*ttp://www.unicode.org/versions/Unicode4.1.0/*>.

Author’s Address

Douglas Crockford
JSON.org
EMail: *douglas@crockford.com*

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

Html markup produced by rfcmarkup 1.66, available from *http://tools.ietf.org/tools/rfcmarkup/*

Index