AS/400e

HTTP Server for AS/400 Web Programming Guide

GC41-5435-04

AS/400e

HTTP Server for AS/400 Web Programming Guide

GC41-5435-04

Note

Before using this information and the product it supports, be sure to read the general information under "Chapter 11. Notices" on page 145.

Fifth Edition (May 2000)

This edition applies to the IBM HTTP Server for AS/400 licensed program (Program 5769-DG1), Version 4 Release 5 Modification 0 and to all subsequent releases and modifications until otherwise indicated in new editions. This edition applies only to reduced instruction set computer (RISC) systems.

This edition replaces GC41-5435-03.

Contents

About HTTP Server for AS/400 Web Programming Guide (GC41-5435)

The web is an interactive medium. For example, it allows users to use search utilities to locate information on a topic, give feedback to a company about its products, and more. The IBM HTTP Server software does not perform these tasks. They are performed by external programs using information passed to them by the server. The *HTTP Server for AS/400 Web Programming Guide* tells you how to write external programs that interact with the IBM HTTP Server for AS/400 product.

The *HTTP Server for AS/400 Webmaster's Guide*, GC41-5434, describes the configuration directives used to set up and control the IBM HTTP Server for AS/400 product.

The *IBM AS/400 Information Center* presents information about the Web server and many other AS/400 products and topics in an electronic, searchable database format. The Information Center offers assistance in setting up and configuring your Web server and publishing a Web site, as well as the advanced functions such as logging and proxy serving. It is available on the Internet at http://publib.boulder.ibm.com/html/as400/infocenter.html or on CD-ROM: *AS/400 Information Center*, SK3T-2027-03

The IBM HTTP Server is IBM's web server, and it is a cross platform product. With the IBM HTTP Server you can serve multimedia objects such as Hypertext Markup Language (HTML) documents to World Wide Web browser clients with your AS/400 system. The IBM HTTP Server for AS/400 is fully HTTP 1.1 compliant.

The IBM HTTP Server for AS/400 (that was introduced in V4R3) replaces the IBM AS/400 Internet Connection Server (ICS) introduced in OS/400 V4R1.

Conventions in this book

Boldface	Indicates the name of an item you need to select, the name of a field, or a string you must enter.
Italics	Indicates book titles or variable information that must be replaced by an actual value.
Bold italics	Indicates a new term.
`Monospace`	Indicates an example, a portion of a file, or a previously entered value.

AS/400 Operations Navigator

AS/400 Operations Navigator is a powerful graphical interface for Windows clients. With AS/400 Operations Navigator, you can manage and administer your AS/400 systems from your Windows desktop.

You can use Operations Navigator to manage communications, printing, database, security, and other system operations. Operations Navigator includes Management Central for managing multiple AS/400 systems centrally.

This new interface has been designed to make you more productive and is the only user interface to new, advanced features of OS/400. Therefore, IBM

recommends that you use AS/400 Operations Navigator, which has online help to guide you. While this interface is being developed, you may still need to use an emulator such as PC5250 to do some of your tasks.

Installing Operations Navigator

To use AS/400 Operations Navigator, you must have Client Access installed on your Windows PC. For help in connecting your Windows PC to your AS/400 system, consult *Client Access Express for Windows - Setup*, SC41-5507-01.

AS/400 Operations Navigator is a separately installable component of Client Access that contains many subcomponents. If you are installing for the first time and you use the **Typical** installation option, the following options are installed by default:

- Operations Navigator base support
- Basic operations (messages, printer output, and printers)

To select the subcomponents that you want to install, select the **Custom** installation option. (After Operations Navigator has been installed, you can add subcomponents by using Client Access Selective Setup.)

After you install Client Access, double-click the **AS400 Operations Navigator** icon on your desktop to access Operations Navigator and create an AS/400 connection.

Prerequisite and related information

Use the AS/400 Information Center as a starting point for your AS/400 information needs. It is available in either of the following ways:

- The Internet at this uniform resource locator (URL) address:
  `http://publib.boulder.ibm.com/html/as400/infocenter.html`
- On CD-ROM: *AS/400 Information Center*, SK3T-2027-03 .

The AS/400 Information Center contains important topics such as logical partitioning, clustering, Java, TCP/IP, Web serving, and secured networks. It also contains Internet links to Web sites such as the AS/400 Online Library and the AS/400 Technical Studio. Included in the Information Center is a link that describes at a high level the differences in information between the Information Center and the Online Library.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other AS/400 documentation, fill out the readers' comment form at the back of this book.

- If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
  - United States and Canada: 1-800-937-3430
  - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use this network ID:
  - IBMMAIL, to IBMMAIL(USIB56RZ)

– RCHCLERK@us.ibm.com

Be sure to include the following:

- The name of the book.
- The publication number of the book.
- The page number or topic to which your comment applies.

Chapter 1. Writing Common Gateway Interface Programs

This chapter discusses the Common Gateway Interface (CGI): What it is, why you might want to use it, and how it works.

Overview of the CGI

CGI is a standard, supported by almost all web servers, that defines how information is exchanged between a web server and an external program (CGI program).

The CGI specification dictates how CGI programs get their input and how they produce any output. CGI programs process data that is received from browser clients. For example, the client fills out a form and sends the information back to the server. Then the server runs the CGI program.

Programs that are called by the server must conform to the server CGI interface in order to run properly. We will describe this in further detail later in this chapter.

The administrator controls which CGI programs the system can run by using the server directives. The server recognizes a URL that contains a request for a CGI program, commonly called a CGI script. Depending on the server directives, the server calls that program on behalf of the client browser.

For more information on CGI, see this URL:

http://www.w3.org/Daemon/User/CGI/Overview.html.

AS/400® supports CGI programs that are written in C++, Rexx, Java®, ILE-C, ILE-RPG, and ILE-COBOL. It also supports multi-thread CGI programs in all of these languages capable of multiple threads. For sample code that can help you with CGI programs, see the AS/400 Web Builder's Workshop. You can find it at the following URL:

http://www.as400.ibm.com/tstudio/workshop/webbuild.htm.

You need to compile programs that are written in programming languages. Compiled programs typically run faster than uncompiled programs. On the other hand, those programs that are written in scripting languages tend to be easier to write, maintain, and debug.

The functions and tasks that CGI programs can perform range from the simple to the very advanced. In general, we call those that perform the simple tasks ***CGI scripts*** because you do not compile them. We often call those that perform complex tasks ***gateway programs***. In this manual, we refer to both types as ***CGI programs***.

Given the wide choice of languages and the variety of functions, the possible uses for CGI programs seem almost endless. How you use them is up to you. Once you understand the CGI specification, you will know how servers pass input to CGI programs and how servers expect output.

There are many uses for CGI programs. Basically, you should design them to handle dynamic information. Dynamic in this context refers to temporary information that is created for a one-time use and not stored anywhere on the web. This information may be a document, an e-mail message, or the results of a conversion program.

For detailed information about AS/400 CGI APIs, see "Chapter 2. Application Programming Interfaces" on page 23.

CGI and Dynamic Documents

There are many types of files that exist on the web. Primarily they fall into one of the following categories:

- Images
- Multimedia
- Programs
- HTML documents

Servers break HTML documents into two distinct types:

- Static
- Dynamic

Static documents exist in non-changing source form on the web server. You should create ***Dynamic documents*** as temporary documents to satisfy a specific, individual request.

Consider the process of "serving", these two types of documents. Responding to requests for static documents is fairly simple. For example, Jill User accesses the Acme web server to get information on the Pro-Expert gas grill. She clicks on Products, then on Grills, and finally on Pro-Expert. Each time Jill clicks on a link, the web browser uses the URL that is attached to the link to request a specific document from the web server. The server responds by sending a copy of the document to Jill's browser.

What if Jill decides that, she wants to search through the information on the Acme web server for all documents that contain information on Acme grills? Such information could consist of news articles, press releases, price listings, and service agreements. This is a more difficult request to process. This is not a request for an existing document. Instead, it is a request for a dynamically generated list of documents that meet certain criteria. This is where CGI comes in.

You can use a CGI program to parse the request and search through the documents on your web server. You can also use it to create a list with hypertext links to each of the documents that contain the specified word or string.

Uses for CGI

HTML allows you to access resources on the Internet by using other protocols that are specified in the URL. Examples of such protocols are mailto, ftp, and news. If you code a link with mailto that is followed by an e-mail address, the link will result in a generic mail form.

What if you wanted your customers to provide specific information, such as how often they use the web? Or how they heard about your company? Rather than using the generic mailto form, you can create a form that asks these questions and more. You can then use a CGI program to interpret the information, include it in an e-mail message, and send it to the appropriate person.

You do not need to limit CGI programs to processing search requests and e-mail. You can use them for a wide variety of purposes. Basically, anytime you want to take input from the reader and generate a response, you can use a CGI program. The input may even be apparent to the reader. For example, many people want to know how many other people have visited their home page. You can create a CGI program that keeps track of the number of requests for your home page. This program can display the new total each time someone links to your home page.

The CGI process

Usually CGI programs are referred to from within HTML documents. In general, the HTML document format defines the environment variables that specify the passing of information. When you design the layout of your HTML document, you must keep in mind how a CGI program might affect the look of your document. Developing the CGI program along with the HTML document will help you avoid many design mistakes.

Overview

The CGI process involves three players: The web browser, the web server, and the CGI program. To exemplify how CGI programs for online forms work, let us assume that the web browser has already requested and obtained an HTML form.

1. The user clicks buttons or enters information in fields, and then clicks on an HTML button to submit the request.
2. The web browser then sends the data to the web server in an encoded format. For example, the data might consist of responses on an HTML form.
3. When the web server receives data, it converts the data to a format compliant with the CGI specification for input and sends it to the CGI program.
4. The CGI program then decodes and processes the data.
5. The system sends this response back to the web server in a form that is compliant with the CGI specification for output.
6. The web server then interprets the response and forwards it to the web browser.

Note: If a CGI application program must change the HTTP server job attributes while processing, the CGI program must restore the attributes to their initial values before the CGI program ends. Failure to restore job attributes that are changed in the CGI program will result in unpredictable responses to future server requests. For example, a CGI program that requires a library in the library list needs to add the library to the library list. The CGI program must remove the library list before the CGI program ends.

The following HTML form illustrates the various types of fields:

Note: The CGIXMP.EXE program referred to in this sample is just an example; it is not shipped with the server product.

```
<HTML>
<HEAD>
<TITLE>CGIXMP Test Case</TITLE>
</HEAD>
<BODY>
<H1>CGI Sample Test Case</H1>
Fill in the following fields and press APPLY.
The values you enter will
be read by the CGIXMP.EXE program and displayed in a simple HTML
form which is generated dynamically by the program.
<P> <HR>
<form method=POST action="/cgi-bin/cgixmp">
<P>
<H3>Checkbox Field</H3>
<P>
<PRE>
<input type="checkbox" name="var1" value="123">
Check to set variable VAR1 to 123<BR>
<input type="checkbox" name="var2" value="XyZ" checked>
Check to set variable VAR2 to XyZ<BR>
</PRE>
<P>
<H3>Radio Button Field</H3>
<P>
<PRE>
<input type="radio" name="var3" value="1">
Select to set variable VAR3 to 1<BR>
<input type="radio" name="var3" value="2">
Select to set variable VAR3 to 2<BR>
<input type="radio" name="var3" value="3" checked>
Select to set variable VAR3 to 3<BR>
<input type="radio" name="var3" value="4">
Select to set variable VAR3 to 4<BR>
</PRE>
<P>
<H3>Single selection List Field</H3>
<P>
<PRE>
Select a value for variable VAR4   <select size=1 name="var4">
<option>0<option>1<option>2<option>3
<option>4<option>5</select>
</PRE>
<P>
<H3>Text Entry Field</H3>
<P>
<PRE>
Enter value for variable VAR5 <input type="text" name="var5"
size=20 maxlength=256 value="TEST value">
</PRE>
<P>
<H3>Multiple selection List Field</H3>
<P>
<PRE>
Select a value for variable VAR6
<select multiple size=2 name="var6">
<option>Ford<option>Chevrolet<option>Chrysler<option>
Ferrari<option>Porsche
</select>
</PRE>
<P>
<H3>Password Field</H3>
<P>
```

```
<PRE>
Enter Password
<input type="password" name="pword" size=10 maxlength=10>
</PRE>
<P>
<H3>Hidden Field</H3>
<P>
<input type="hidden" name="hidden" value="Text not shown on form...">
<P>
<PRE>
<input type="submit" name="pushbutton" value="Apply">
<input type="reset" name="pushbutton" value="Reset">
<HR>
</PRE>
</FORM>
</BODY>
</HTML>
```

Sending Information to the Server

When you fill out a form, the web browser sends the request to the server in a format that is described as ***URL-encoded***. The web browser also performs this function whenever you enter a phrase in a search field and click on the submission button. In URL-encoded information:

- The URL starts with the URL of the processing program.
- A question mark (?) precedes attached data, such as name=value information from a form, which the system appends to the URL.
- A plus sign (+) represents spaces.
- A percent sign (%) that is followed by the American National Standard Code for Information Interchange (ASCII) hexadecimal equivalent of the symbol represents special characters, such as a period (.) or slash (/).
- An ampersand (&) separates fields and sends multiple values for a field such as check boxes.

Note: The method attribute specifies how the server sends the form information to the program. You use the GET and POST methods in the HTML file to process forms. The GET method sends the information through environment variables. You will see the information in the URL after the ″?″ character. The POST method passes the data through standard input.

The main advantage of using the GET method is that you can access the CGI program with a query without using a form. In other words, you can create canned queries that pass parameters to the program. For example, if you want to send the previous query to the program directly, you can do the following:

```
<A HREF="/cgi-bin/program.pgm?Name=John&LName=Richard&user=Smith%Company=IBM">
YourCGI Program</a>
```

The main advantage to the POST method is that the query length can be unlimited so you do not have to worry about the client or server truncating data. The query string of the GET method cannot exceed 4 KB.

Data Conversions on CGI Input and Output

The server can perform ASCII to EBCDIC conversions before sending data to CGI programs. This is because the Internet is an ASCII world and the AS/400 is primarily an extended binary-coded decimal interchange code (EBCDIC) world. The server can also perform EBCDIC to ASCII conversions before sending data

back to the browser. You must provide data to a CGI program through environment variables and standard-input (stdin). HTTP and HTML specifications allow you to tag text data with a character set (charset parameter on the Content-Type header). However, this practice is not widely in use today (although technically required for HTTP1.0/1.1 compliance). According to this specification, text data that is not tagged can be assumed to be in the default character set ISO-8859-1 (US-ASCII). AS/400 correlates this character set with ASCII CCSID 819.

There are basically three different ways the server can process the input to and output from your CGI program. You can configure the server to control which mode is used by specifying an overall server directive (CGIConvMode) or an optional parameter on the Exec or Post-Script script directives:

```
CGIConvMode Mode
Exec request-template program-path [server-IP-address or hostname] [Mode]
Post-Script program_path_and_name [server-IP-address or hostname] [Mode]
```

Where *Mode* is one of the following:

%%MIXED%% or **%%MIXED/MIXED%%** This is the default.

%%EBCDIC%% or **%%EBCDIC/MIXED%%**

%%EBCDIC/EBCDIC%%

%%BINARY%% or **%%BINARY/MIXED%%**

%%BINARY/EBCDIC%%

%%BINARY/BINARY%%

%%EBCDIC_JCD%% or **%%EBCDIC_JCD/MIXED%%**

%%EBCDIC_JCD/EBCDIC%%

The CGIMode can be thought of as 2 logical pieces. The input mode and output mode. They are separated by the "/". If only the input mode is provided, the output mode is defaulted to MIXED for compatibility.

In addition, the system provides the following CGI environment variables to the CGI program:

- CGI_MODE - which input conversion mode the server is using (%%MIXED%%, %%EBCDIC%%, %%BINARY%%, or %%EBCDIC_JCD%%).
- CGI_ASCII_CCSID - from which ASCII CCSID was used to convert the data
- CGI_EBCDIC_CCSID - which EBCDIC CCSID the data was converted into
- CGI_OUTPUT_MODE - which output conversion mode the server is using (%%MIXED%%, %%EBCDIC%%, or %%BINARY%%)

The following section explains CGI input conversion modes in more detail.

CGI Input Conversion Modes

MIXED

This mode is the default mode of operation for the server. The system converts values for CGI environment variables to EBCDIC CCSID 37, including QUERY_STRING. The system converts stdin data to the CCSID of the job. However, the system still represents the encoded characters "%xx" by the ASCII 819 octet. This requires the CGI program to convert these further into EBCDIC to process the data. For more information, see symptom, *Special characters are not being converted or handled as expected* in "Chapter 10. Troubleshooting your CGI programs" on page 139.

EBCDIC

In this mode, the server will convert everything into the EBCDIC CCSID of the job. The server checks the Entity bodies for a charset tag. If found, the server will convert the corresponding ASCII CCSID to the EBCDIC CCSID of the job. If the server does not find a charset tag, it uses the value of the DefaultNetCCSID configuration directive as the conversion CCSID. In addition, the system converts escaped octets from ASCII to EBCDIC, eliminating the need to perform this conversion in the CGI program.

BINARY

In this mode, the server processes environment variables (except QUERY_STRING) the same way as EBCDIC mode. The server performs no conversions on either QUERY_STRING or stdin data.

EBCDIC_JCD

Japanese browsers can potentially send data in one of three code pages, JIS (ISO-2022-JP), S-JIS (PC-Windows), or EUC (UNIX). In this mode, the server uses a well-known JCD utility to determine which codepage to use (if not explicitly specified by a charset tag) to convert stdin data.

Table 1 summarizes the type of conversion that is performed by the server for each CGI mode.

Table 1. Conversion action for text in CGI Stdin.

CGI_MODE	Conversion	Stdin encoding	Environment variables	Query_String encoding	argv encoding
%%BINARY%%	None	No conversion	FsCCSID	No conversion	No conversion
%%EBCDIC%%	NetCCSID to FsCCSID	FsCCSID	FsCCSID	FsCCSID	FsCCSID
%%EBCDIC%% - with charset tag received	Calculate target EBCDIC CCSID based on received ASCII charset tag	EBCDIC equivalent of received charset	FsCCSID	FsCCSID	FsCCSID
%%EBCDIC_JCD%%	Detect input based on received data. Convert data to FsCCSID	FsCCSID	FsCCSID	Detect ASCII input based on received data. Convert data to FsCCSID	Detect ASCII input based on received data. Convert data to FsCCSID
%%MIXED%% (Compatability mode)	NetCCSID to FsCCSID (receive charset tag is ignored)	FsCCSID with ASCII escape sequences	CCSID 37	CCSID 37 with ASCII escape sequences	CCSID37 with ASCII escape sequences

DBCS Considerations

URL-encoded forms containing DBCS data could contain ASCII octets that represent parts of DBCS characters. The server can only convert non-encoded character data. This means that it must un-encode the double-byte character set (DBCS) stdin and QUERY_STRING data before performing the conversion. In addition, it has to reassemble and re-encode the resulting EBCDIC representation before passing it to the CGI program. Because of this extra processing, CGI programs that you write to handle DBCS data may choose to receive the data as BINARY and perform all conversions to streamline the entire process.

Using the EBCDIC_JCD mode: The EBCDIC_JCD mode determines what character set is being used by the browser for a given request. This mode is also used to automatically adjust the ASCII/EBCDIC code conversions used by the web server as the request is processed.

After auto detection, the %%EBCDIC_JCD%% mode converts the stdin and QUERY_STRING data from the detected network CCSID into the correct EBCDIC CCSID for Japanese. The default conversions configured for the server instance are overridden. The DefaultFsCCSID directive or the -fsccsid startup parameter specifies the default conversions. The startup FsCCSID must be a Japanese CCSID (5026 or 5035).

The possible detected network code page is Shift JIS, eucJP, and ISO-2022-JP. The following are the associated CCSIDs for each code page:

```
Shift JIS   (See note 1)
=========
CCSID 932:  IBM PC (old JIS sequence, OS/2 J3.X/4.0, IBM Windows J3.1)
CCSID 942:  IBM PC (old JIS sequence, OS/2 J3.X/4.0)
CCSID 943:  MS Shift JIS (new JIS sequence, OS/2 J4.0
            MS Windows J3.1/95/NT)

eucJP
=====
CCSID 5050:  Extended UNIX Code (Japanese)


ISO-2022-JP  (See note 2)
===========
CCSID 5052:  Subset of RFC 1468 ISO-2022-JP (JIS X 0201 Roman and
             JIS X 0208-1983) plus JIS X 0201 Katakana.

CCSID 5054:  Subset of RFC 1468 ISO-20220JP (ASCII and JIS X 0208-1983)
             plus JIS X 0201 Katakana.
```

The detected network CCSID is available to the CGI program. The CCSID is stored in the CGI_ASCII_CCSID environment variable. When JCD can not detect, the default code conversion is done as configured (between NetCCSID and FsCCSID). (See note 3.)

Since the code page of Stdin and QUERY_STRING are encoded according to the web client's outbound code page, we recommend using the following configuration value combinations when you use the %%EBCDIC_JCD%% mode.

```
Startup FsCCSID           Startup NetCCSID          Description
---------------           ----------------          ---------------------
5026/5035 (See note 4) <->  943                     Default:  MS Shift JIS
5026/5035 (See note 4) <->  942                     Default:  IBM PC
5026/5035 (See note 4) <->  5052/5054               Default:  ISO-2022-JP
```

Using CCSID 5050(eucJP) for the startup NetCCSID, is not recommended. When 5050 is specified for the startup NetCCSID, the default code conversion is done between FsCCSID and 5050. This means that if JCD cannot detect a code page, JCD returns 5050 as the default network CCSID. Most browser's use a default outbound code page of Shift JIS or ISO-2022-JP, not eucJP.

If the web client sends a charset tag, JCD gives priority to the charset tag. Stdout function is the same. If the charset/ccsid tag is specified in the Content-Type field, stdout gives priority to charset/ccsid tag. Stdout also ignores the JCD detected network CCSID. (See note 5.)

Notes:

1. If startup NetCCSID is 932 or 942, detected network, Shift JIS's CCSID is the same as startup NetCCSID. Otherwise, Shift JIS's CCSID is 943.

Startup NetCCSID	Shift JIS (JCD detected CCSID)
932	932
942	942
943	943
5052	943
5054	943
5050	943

2. Netscape Navigator 3.x sends the alphanumeric characters by using JIS X 0201 Roman escape sequence (CCSID 5052) for ISO-2022-JP. Netscape Communicator 4.x sends the alphanumeric characters by using ASCII escape sequence (CCSID 5054) for ISO-2022-JP.
3. JCD function has the capability to detect EUC and SBCS Katakana, but it is difficult to detect them. IBM® recommends that you do not use SBCS Katakana and EUC in CGI.
4. CCSID 5026 assigns lowercase alphabet characters on special code point. This often causes a problem with lowercase alphabet characters. To avoid this problem, do one of the following:
   - Do not use lowercase alphabet literals in CGI programs if the FsCCSID is 5026.
   - Use CCSID 5035 for FsCCSID.
   - Use the Charset/CCSID tag as illustrated in the following excerpt of a CGI program:

     ```
     main(){
       printf("Content-Type: text/html; Charset=ISO-2022-JP\n\n");
       ...
     }
     ```

   - Do the code conversions in the CGI program. The following sample C program converts the literals into CCSID 930 (the equivalent to CCSID 5026)

     ```
     main(){
       printf("Content-Type: text/html\n\n);

     #pragama convert(930)
        printf("<html>");
        printf("This is katakana code page\n");
     #pragama convert(0)
          ...
     }
     ```

5. When the web client sends a charset tag, the network CCSID becomes the ASCII CCSID associated with Multipurpose Internet Mail Extensions (MIME) charset header. The charset tag ignores the JCD detected CCSID. When the Charset/ccsid tag is in the Content-Type header generated by the CGI program, the JCD-detected CCSID is ignored by this charset/ccsid. Stdout will not perform a conversion if the charset is the same as the MIME's charset. Stdout will not perform a conversion if the CCSID is ASCII. Stdout will perform code conversion if the CCSID is EBCDIC. Because the environment variables and stdin are already stored in FsCCSID, ensure that you are consistent between the FsCCSID and the Content-Type header's CCSID.

CGI Output Conversion Modes

The CgiConv mode includes an output mode. This section explains CGI output conversion modes in more detail.

%%MIXED%%

In this mode HTTP header output is in CCSID 37. However, the escape sequence must be the EBCDIC representative of the ASCII code point for the 2 characters following the ″%″ in the escape sequence. An example of a HTTP header that may contain escape sequences is the Location header.

%%EBCDIC%%

In this mode HTTP header output is in CCSID 37. However, the escape sequence must be the EBCDIC representative of the EBCDIC code point for the 2 characters following the ″%″ in the escape sequence. An example of a HTTP header that may contain escape sequences is the Location header.

%%BINARY%%

In this mode HTTP header output is in CCSID 819 with the escape sequences also being the ASCII representative of the ASCII code point. An example of a HTTP header that may contain escape sequences is the Location header.

For HTTP body standard-output (stdout) data that is sent from the CGI program, the server recognizes and uses the charset or CCSID parameter on the text/* Content-Types. If you specify ASCII, the server performs no conversions on the data. Otherwise, the system uses the Content-Type value instead of the DefaultFsCCSID on conversions back to the browser. The system sets an appropriate charset tag for all text/* Content-Types that it sends back to the browser. The exception to this is %%MIXED%%, %%MIXED/MIXED%%, %%BINARY/BINARY%% modes and when the charset or CCSID parameter is set to 65535.

Table 2 summarizes the type of conversion that is performed and the charset tag that is returned to the browser by the server.

Table 2. Conversion action and charset tag generation for text in CGI Stdout.

CGI Stdout CCSID/Charset in HTTP header	Conversion action	Server reply charset tag
EBCDIC CCSID/Charset	Calculate EBCDIC to ASCII conversion based on supplied EBCDIC CCSID/Charset	Calculated ASCII charset
ASCII CCSID/Charset	No conversion	Stdout CCSID/Charset as Charset
65535	No conversion	None
None (%%BINARY%%, %%BINARY/MIXED%%, or %%BINARY/EBCDIC%%)	Default Conversion - FsCCSID to NetCCSID	NetCCSID as charset
None (%%BINARY/BINARY%%)	No Conversion	None
None (%%EBCDIC%%, %%EBCDIC/MIXED%%, or %%EBCDIC/EBCDIC%%)	Default Conversion - FsCCSID to NetCCSID	NetCCSID as charset
None (%%EBCDIC%%, %%EBCDIC/MIXED%%, or %%EBCDIC/EBCDIC%% with charset tag received on HTTP request)	Use inverse of conversion calculated for stdin	Charset as received on HTTP request
None (%%EBCDIC_JCD%%, %%EBCDIC_JCD/MIXED%%, or %%EBCDIC_JCD/EBCDIC%%)	Default Conversion - FsCCSID to NetCCSID	NetCCSID as charset

Table 2. Conversion action and charset tag generation for text in CGI Stdout. (continued)

CGI Stdout CCSID/Charset in HTTP header	Conversion action	Server reply charset tag
None (%%MIXED%% or %%MIXED/MIXED%%)	Default Conversion - FsCCSID to NetCCSID	None (compatibility mode)
Invalid	CGI error 500 generated by server	

The server also sets an environment variable CGI_OUTPUT_MODE to reflect the setting for the CGI output mode. It contains the CGI output conversion mode the server is using for this request. Valid values are %%EBCDIC%%, %%MIXED%%, or %%BINARY%%. The program can use this information to determine what conversion, if any, the server performs on CGI output.

Returning Output from the Server

When the CGI program is finished, it passes the resulting response to the server by using standard output (stdout). The server interprets the response and sends it to the browser.

A CGI program writes a CGI header that is followed by an entity body to standard output. The CGI header is the information that describes the data in the entity body. The entity body is the data that the server sends to the client. A single newline character always ends the CGI header. The newline character for ILE/C is \n. For ILE/RPG or ILE/COBOL, it is hexadecimal '15'. The following are some examples of Content-Type headers:

```
Content-Type: text/html\n\n
Content-Type: text/html; charset=iso-8859-2\n\n
```

If the response is a static document, the CGI program returns either the URL of the document using the CGI Location header or returns a Status header. The CGI program does not have an entity body when using the Location header. If the host name is the local host, the HTTP server will retrieve the specified document that the CGI program sent. It will then send a copy to the web browser. If the host name is not the local host, the HTTP processes it as a redirect to the web browser. For example:

```
Location: http://www.acme.com/products.html\n\n
```

The Status header should have a Content_Type: and a Status in the CGI header. When Status is in the CGI header, an entity body should be sent with the data to be returned by the server. The entity body data contains information that the CGI program provides to a client for error processing. The Status line is the Status with an HTTP 3 digit status code and a string of alphanumeric characters (A-Z, a-z, 0-9 and space). The HTTP status code must be a valid 3 digit number from the HTTP/1.1 specification.

Note: The newline character \n ends the CGI header.

```
CONTENT-TYPE: text/html\n
Status:  600 Invalid data\n
\n
<html><head><title>Invalid data</title>
</head><body>
<h1>Invalid data typed</h1>
<br><pre>
The data entered must be valid numeric digits for id number
<br></pre>
</body></html>
```

How CGI Programs Work

Most CGI programs include the following three stages:

- Parsing CGI programs
- Data manipulation within a CGI program
- Response generation by a CGI program

Parsing

Parsing is the first stage of a CGI program. In this stage, the program takes the data from QUERY_STRING environment variable, command line arguments using argv() or standard input. When the method is GET, the system reads the data from the QUERY_STRING environment variable or command line arguments by using argv(). There is no way to determine the length of data in QUERY_STRING. The system encodes the QUERY_STRING data in the request header.

An example of data read in the QUERY_STRING variable (%%MIXED%% mode):

```
NAME=Eugene+T%2E+Fox&ADDR=etfox%40ibm.net&INTEREST=RCO
```

Parsing breaks the fields at the ampersands and decodes the ASCII hexadecimal characters. The results look like this:

```
NAME=Eugene T. Fox
ADDR=etfox@ibm.net
INTEREST=RCO
```

You can use the QtmhCvtDb API to parse the information into a structure. The CGI program can refer to the structure fields. If using %%MIXED%% input mode, the "%xx" encoding values are in ASCII and must be converted into the "%xx" EBCDIC encoding values before calling QtmhCvtDb. If using %%EBCDIC%% mode, the server will do this conversion for you. The system converts ASCII "%xx" first to the ASCII character and then to the EBCDIC character. Ultimately, the system sets the EBCDIC character to the "%xx" in the EBCDIC CCSID. For code samples, use the following URL to the AS/400 web site: **http://www.as400.ibm.com/tstudio/index.htm.**

When the method is post, the system reads the data from standard input. Before the CGI attempts to read standard input, it must check environment variables REQUEST_METHOD and CONTENT_LENGTH. Read standard input only when the REQUEST_METHOD is POST. The read must specify no more than CONTENT_LENGTH bytes. Attempts to specify more than CONTENT_LENGTH bytes on reading standard input are not defined.

Data manipulation

Data manipulation is the second stage of a CGI program. In this stage, the program takes the parsed data and performs the appropriate action. For example, a CGI program designed to process an application form might perform one of the following functions:

1. Take the input from the parsing stage
2. Convert abbreviations into more meaningful information
3. Plug the information into an e-mail template
4. Use SNDDST to send the e-mail.

Response generation

Response generation is the final stage of a CGI program. In this stage, the program formulates its response to the web server, which forwards it to the browser. The response contains MIME headers that vary depending on the type of response.

With a search, the response might be the URLs of all the documents that met the search value. With a request that results in e-mail, the response might be a message that confirms that the system actually sent the e-mail.

Environment variables

Before you begin writing your CGI program, you need to understand the format in which the server will pass the data. The server receives the URL-encoded information and, depending on the type of request, passes the information to the CGI program. The server does this by using environment variables, command line arguments, or standard input.

A CGI application program should be able to handle a NULL value when getting an environment variable. For example, when the CGI program is trying to do a getenv("CONTENT_LENGTH") and the method is GET, the value would be returned NULL. This is because CONTENT_LENGTH is only defined in method POST (to describe the length of standard input).

The system supports the following environment variables:

AUTH_TYPE
If the server supports client authentication and the script is a protected script, this environment variable contains the method that is used to authenticate the client. For example:

```
Basic
```

CGI_ASCII_CCSID
Contains the ASCII CCSID the server used when converting CGI input data. If the server did not perform any conversion, (for example, in %%BINARY%% mode), the server sets this value to the DefaultNetCCSID configuration directive value.

CGI_MODE
Contains the CGI conversion mode the server is using for this request. Valid values are %%EBCDIC%%, %%MIXED%%, %%BINARY%%, or %%EBCDIC_JCD%%. The program can use this information to determine what conversion, if any, was performed by the server on CGI input data and what format that data is currently in.

CGI_EBCDIC_CCSID
Contains the EBCDIC CCSID under which the current server job is running (DefaultFsCCSID configuration directive). It also represents the current job CCSID that is used during server conversion (if any) of CGI input data.

CONTENT_LENGTH
When the method of POST is used to send information, this variable contains the number of characters. Servers typically do not send an end-of-file flag when they forward the information by using stdin. If needed, you can use the CONTENT_LENGTH value to determine the end of the input string. For example:

```
7034
```

CONTENT_TYPE
When information is sent with the method of POST, this variable contains the type of data included. You can create your own content type in the server configuration file and map it to a viewer. For example:

```
Application/x-www-form-urlencoded
```

GATEWAY_INTERFACE
Contains the version of CGI that the server is using. For example:

```
CGI/1.1
```

HTTP_ACCEPT
Contains the list of MIME types the browser accepts. For example:

```
text/html
```

HTTP_USER_AGENT
Contains the name of your browser (web client). It includes the name and version of the browser, requests that are made through a proxy, and other information. For example:

```
Netscape Navigator dll /v3.0
```

IBM_CCSID_VALUE
The CCSID under which the current server job is running.

PATH_INFO
Contains the additional path information as sent by the web browser. For example:

```
/ballyhoo
```

PATH_TRANSLATED
Contains the decoded or translated version of the path information that is contained in PATH_INFO, which takes the path and does any virtual-to-physical mapping to it. For example:

```
/wwwhome/ballyhoo
```

QUERY_STRING
When information is sent using a method of GET, this variable contains the information in a query that follows the ?. The string is coded in the standard URL format of changing spaces to "+" and encoding special characters with "%xx" hexadecimal encoding. The CGI program must decode this information. For example:

```
NAME=Eugene+T%2E+Fox&ADDR=etfox%7Cibm.net&INTEREST=xyz
```

REMOTE_ADDR
Contains the IP address of the remote host (web browser) that is making the request, if available. For example:

```
9.23.06.8
```

REMOTE_HOST
Contains the host name of the web browser that is making the request, if available. For example:

```
raleigh.ibm.com
```

REMOTE_IDENT
Contains the user ID of the remote user. For example:

```
Jillx
```

REMOTE_USER
If you have a protected script and the server supports client authentication, this environment variable contains the user name that is passed for authentication. For example:

```
Jill
```

REQUEST_METHOD
Contains the method (as specified with the METHOD attribute in an HTML form) that is used to send the request. For example:

```
GET or POST
```

SCRIPT_NAME
A virtual path to the program being run. Use this for self-referring URLs.

SERVER_NAME
Contains the server host name or IP address of the server. For example:

```
www.ibm.com
```

SERVER_PORT
Contains the port number to which the client request was sent. For example:

```
80
```

SERVER_PROTOCOL
Contains the name and version of the information protocol that is used to make the request. For example:

```
HTTP/1.0
```

SERVER_SOFTWARE
Contains the name and version of the information server software that is answering the request. For example:

```
IBM-Secure-ICS/AS/400 Secure HTTP Server
```

Requests from Standard Search (ISINDEX) Documents

ISINDEX is an HTML tag that identifies the document as a standard search document and causes the browser to automatically generate an entry field. When information is sent from an ISINDEX document, the server takes the appended data (the information following the ?), breaks it at the pluses (+), and sends the data to the CGI program as command line arguments (argv). For example:

```
<ISINDEX>
```

Note: It is possible to write CGI scripts that display all environment variables. At times these variables may include sensitive data such as user IDs and passwords for various products. Consequently, you must be careful about displaying environment variables in your CGI scripts, and you must be careful about who has access to them.

Passing SSL Environment Variables to a CGI Program

You can use the following SSL-related environment variables in CGI programs.

HTTPS
Returns ON if the system has completed an SSL handshake. It returns OFF if the exchange of signals to set up communications between two modems has failed. For example:

```
OFF
```

HTTPS_CLIENT_CERT
The entire certificate passed to the server from the client browser when SSL Client authentication is enabled. The format of the certificate is a BASE64 encoded string that represents the DER format of the X.509 certificate.

HTTPS_CLIENT_CERT_COUNTRY
The Country Code from the client certificate's distinguished name. For example:

```
US
```

HTTPS_CLIENT_CERT_COMMON_NAME
The Common Name from the client certificate's distinguished name. For example:

```
John Smith
```

HTTPS_CLIENT_CERT_ISSUER_COMMON_NAME
The Common Name of the Certificate Authority that issued the client's certificate. For example:

```
Digital ID
```

HTTPS_CLIENT_CERT_ISSUER_COUNTRY
The Country code of the Certificate Authority that issued the client's certificate. For example:

```
US
```

HTTPS_CLIENT_CERT_ISSUER_LOCALITY
The Locality of the Certificate Authority that issued the client's certificate. For example:

```
50265
```

HTTPS_CLIENT_CERT_ISSUER_STATE_OR_PROVINCE
The State or Province of the Certificate Authority that issued the client's certificate. For example:

```
Iowa
```

HTTPS_CLIENT_CERT_ISSUER_ORG_UNIT
The Organizational Unit of the Certificate Authority that issued the client's certificate. For example:

```
Department of Client Certificates
```

HTTPS_CLIENT_CERT_ISSUER_ORGANIZATION
The Organization name of the Certificate Authority that issued the client's certificate. For example:

```
Roadrunner CA
```

HTTPS_CLIENT_CERT_LEN
The Length of the certificate passed in HTTPS_CLIENT_CERT.

HTTPS_CLIENT_CERT_LOCALITY
The Locality (zip code in the US) from the client certificate's distinguished name. For example:

```
55901
```

HTTPS_CLIENT_CERT_SERIAL_NUM
The serial number assigned by the issuing Certificate Authority. For example:

```
92787829
```

HTTPS_CLIENT_CERT_ORG_UNIT
The Organizational Unit name from the client certificate's distinguished name. For example:

```
Department of Coyote products
```

HTTPS_CLIENT_CERT_ORGANIZATION
The Organization name from the client certificate's distinguished name. For example:

```
Acme Corporation
```

HTTPS_CLIENT_CERT_STATE_OR_PROVINCE
The State or Province from the client certificate's distinguished name. For example:

```
Minnesota
```

HTTPS_KEYSIZE
Returns the number of bits in the session key that is established by SSL after a completed exchange of signals to set up communications between two modems. This value is blank if HTTPS=OFF. For example:

```
512
```

Examples of key sizes are Export {40} or {128}.

HTTPS_PORT
If a valid security product is installed and the SSLMode directive is SSLMode=ON, this environment variable contains the SSL port number the server is listening on. For example:

```
443
```

CGI Programs and AS/400 Activation Groups

The following section is intended to give a brief overview of AS/400 Activation Groups.

Note: It is very important to become familiar with the details of activation groups prior to developing or porting a CGI application that will use this support.

AS/400 Activation Groups

Program activation is the process that is used to prepare a program to run. The system must activate AS/400 ILE programs before they can be run. Program activation includes the allocation and initialization of static storage for the program in addition to completing the binding of programs to service programs.

Program activation is not an AS/400 unique concept. All modern computer operating systems must perform program initialization and load. What is unique to AS/400 is the concept of Activation Groups. All ILE programs and service programs are activated within an activation group. This substructure contains the resources necessary to run the program. The resources that are contained and are managed with an activation group include:

- Static and automatic program variables
- Dynamic storage
- Temporary data management resources (For example, open files and SQL cursors)
- Certain types of exception handlers and ending procedures

Run-time creation of ILE activation groups is controlled by specifying an activation group attribute when your program or service program is created. The attribute is specified by using the ACTGRP parameter on the CRTPGM or CRTSRVPGM command. The valid options for this attribute include user-named, *NEW, and *CALLER. The following is a brief description of these options:

user-named - A named activation group allows you to manage a collection of ILE programs and ILE service programs as one application. The activation group is created when it is first needed. All programs and service programs that specify the same activation group name use it then.

***NEW**- The name for this activation group is selected by ILE and will always be unique. System-named activation groups are always deleted when the high level language returns.

***CALLER** - Specifying *CALLER causes the ILE program or service program to be activated within the activation group of the calling program. A new activation group is never created with this attribute.

*NEW is the standard behavior that can be expected on other systems such as UNIX®.

Notes:

1. When you create a persistent CGI program, you must specify a named activation group.
2. CGI programs that are not persistent should not refer to job-level scoped resources.

For additional information about activation groups see, *ILE Concepts*, SC41-5606 book.

CGI Considerations

There are advantages to running CGI programs in either a user-named or *CALLER activation group. The performance overhead associated with activating a CGI every time that is requested can be drastically reduced. It is important to understand that because the system does not delete user-named activation groups, normal high level language end verbs cannot provide complete end processing. For example, the system will not close open files, and the system will not return the static and heap storage that are allocated by a program. The program must manage these resources explicitly. This will be especially important when moving CGI programs that rely on end processing to function properly.

Note: When you activate multi-threaded CGI on your web server, you get multiple thread support for your CGI application Your CGI application must end all of its threads before returning to the server. When using multi-thread capable CGI, you need to put the CGI program in a new or named activation group.

The following section shows examples which will work fine running in a *NEW activation group, however will cause problems if run in a user-named or *CALLER activation group.

Activation Group Problem Examples

Note

The following examples are **not** general CGI programming examples. For general CGI programming examples, see "Chapter 6. Sample programs (in Java, C, and RPG)" on page 89.

In the following example a CGI program when run in a *NEW activation group, would write Hello World to the browser. What is important to understand is that this application is taking advantage of job end processing to delete the stdio buffers that are used to buffer the stdout data.

You could build the following CGI program to run in either a user-named or *CALLER activation group. In such an instance, the server will not process the

information that was written to stdout. This will cause the web browser to display a ″Document Contains No Data″ error message. Another application could run again in the same activation group that properly erased stdout. In this instance, the data that has been buffered from previous calls would be sent.

```
#include <stdio.h>
void main(void) {

    /************************************************/
    /* Write header information.                    */
    /************************************************/
    printf("Content-type: text/html\n\n");

    /************************************************/
    /* Write header information.                    */
    /************************************************/
    printf("Hello World\n");

}
```

End processing may not erase stdio buffers so the application must erase the stdout with a fflush(stdout) call. The following example will work regardless of the activation group specification:

```
#include <stdio.h>
void main(void) {

    /************************************************/
    /* Write header information.                    */
    /************************************************/
    printf("Content-type: text/html\n\n");

    /************************************************/
    /* Write header information.                    */
    /************************************************/
    printf("Hello World\n");

    /*----------------------------------------------*/
    /* FIX: Flush stdout.                           */
    /*----------------------------------------------*/
    fflush(stdout);

}
```

When run in a *NEW activation group, this example CGI would read CONTENT_LENGTH bytes of data from stdin and write this back out to stdout. The system has allocated the buffer that is used to hold the data with a malloc. Like the example that is previously shown, this application is relying on several aspects of job end processing to function properly.

If this CGI program were built to run in either a user-named or *CALLER activation group, the following problems would occur:

- As with the simple example that is previously shown, the application is not erasing stdout. This would cause the web browser to display a ″Document Contains No Data″ error message. You could run another application again in the same activation group that properly erased stdout. This would send the data that has been buffered from previous calls.
- Stdin is buffered similar to stdout. If the contents of stdin are not erased, the stdin data on the second and all following calls of the CGI program will be unpredictable and the contents may at times contain information from subsequent requests.

- The heap storage allocated using malloc is not being freed. Over time, a memory leak error like this could use significant amounts of memory. This is a common application error that only surfaces when the application is not running in a *NEW activation group.

```
/*********************************************************************/
/*                                                                   */
/* CGI Example program.                                              */
/*                                                                   */
/*********************************************************************/

#include
void main(void)
{

  char* stdinBuffer;
  char* contentLength;
  int   numBytes;
  int   bytesRead;
  FILE* pStdin;

  /*******************************************************************/
  /* Write the header.                                               */
  /*******************************************************************/
  printf("Content-type: text/html\n\n");

  /*******************************************************************/
  /* Get the length of data on stdin.                                */
  /*******************************************************************/
  contentLength = getenv("CONTENT_LENGTH");

  if (contentLength != NULL) {

    /*****************************************************************/
    /* Allocate storage and clear the storage to hold the data.      */
    /*****************************************************************/
    numBytes = atoi(contentLength);
    stdinBuffer = (char*)malloc(numBytes+1);
    if ( stdinBuffer )
      memset(stdinBuffer, 0x00, numBytes+1);

    /*****************************************************************/
    /* Read the data from stdin and write back to stdout.            */
    /*****************************************************************/
    bytesRead = fread(stdinBuffer, 1, numBytes, pStdin);
    stdinBufferþbytesRead+1þ = '\0';
    printf("%s", stdinBuffer);

  } else
    printf("Error getting content length\n");

  return;

}
```

The following example shows the changes that would be required to this application to allow it to run in a user-named or *CALLER activation group:

```
/*********************************************************************/
/*                                                                   */
/* CGI Example program with changes to support user-named            */
/* and *CALLER ACTGRP.                                               */
/*                                                                   */
/*********************************************************************/

#include
void main(void)
```

```
{

  char* stdinBuffer;
  char* contentLength;
  int   numBytes;
  int   bytesRead;
  FILE* pStdin;

  /*********************************************************************/
  /* Write the header.                                                 */
  /*********************************************************************/
  printf("Content-type: text/html\n\n");

  /*********************************************************************/
  /* Get the length of data on stdin.                                  */
  /*********************************************************************/
  contentLength = getenv("CONTENT_LENGTH");

  if (contentLength != NULL) {

    /*******************************************************************/
    /* Allocate storage and clear the storage to hold the data.        */
    /*******************************************************************/
    numBytes = atoi(contentLength);
    stdinBuffer = (char*)malloc(numBytes+1);
    if ( stdinBuffer )
      memset(stdinBuffer, 0x00, numBytes+1);

    /*-----------------------------------------------------------------*/
    /* FIX 2: Reset stdin buffers.                                     */
    /*-----------------------------------------------------------------*/
    pStdin = freopen("", "r", stdin);

    /*******************************************************************/
    /* Read the data from stdin and write back to stdout.              */
    /*******************************************************************/
    bytesRead = fread(stdinBuffer, 1, numBytes, pStdin);
    stdinBuffer[bytesRead+1] = '\0';
    printf("%s", stdinBuffer);

    /*-----------------------------------------------------------------*/
    /* FIX 3: Free allocated memory.                                   */
    /*-----------------------------------------------------------------*/
    free(stdinBuffer);

  } else
    printf("Error getting content length\n");

  /*-------------------------------------------------------------------*/
  /* FIX 1: Flush stdout.                                              */
  /*-------------------------------------------------------------------*/
  fflush(stdout);
  return;

}
```

Chapter 2. Application Programming Interfaces

This chapter includes detailed information on application programming interfaces (APIs) used with the IBM HTTP Server for AS/400.

AS/400 supports these APIs in C++, Java, Rexx, ILE C, ILE COBOL, and ILE RPG programming languages. Although all APIs are supported in all of these languages, most C CGI applications will only need to use QtmhCvtDB, QzhbCgiParse, or QzhbCgiUtils. This is because ANSI C can work with stdin, stdout, and environment variables directly. ILE C CGI applications use ANSI C function calls to work with stdin, stdout, environment variables, and string functions for parsing stdin and environment variable data.

APIs for CGI applications

To use these APIs in a CGI application, you must bind the CGI program to *SRVPGM QZHBCGI in library QHTTPSVR. ILE C programs must include header file QSYSINC/H(QZHBCGI). AS/400 CGI application programs must be written and compiled in Integrated Language Environment® (ILE)/C, ILE/RPG, and ILE/COBOL.

Get Environment Variable (QtmhGetEnv) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Length of response	Output	Binary(4)
4	Request variable	Input	Char(*)
5	Length of request variable	Input	Binary(4)
6	Error Code	I/O	CHAR(*)

The *QtmhGetEnv* API allows you to get the value set by the server for a particular HTTP environment variable.

Required parameter group

Receiver variable
OUTPUT:CHAR(*)

The output variable that contains the value set by the server for the requested environment variable. In CGI input mode %%MIXED%%, this value will be in CCSID 37; otherwise, it will be in the CCSID of the current job. Note that the QUERY_STRING in %%BINARY%% mode is not converted by the server.

Length of receiver variable
INPUT:BINARY(4)

The input variable containing the length of the space provided to receive the environment variable's value.

Length of response
OUTPUT:BINARY(4)

The output variable that contains the length of the environment variable's value. When the API is unable to determine the value for the requested environment variable, the length of the environment variable value is set to zero. When the size required for the environment variable value is larger than the length of the receiver variable, the size required to receive the value is returned.

Request variable
INPUT:CHAR(*)

The input variable containing the desired environment variable's name.

Length of request variable
INPUT:BINARY(4)

The input variable containing the length of the desired environment variable's name.

Error Code
I/O:CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Error messages

CPF24B4 E
Severe Error while addressing parameter list.

CPF3C17 E
Error occurred with input data parameter.

CPF3C19 E
Error occurred with receiver variable specified.

CPF3CF1 E
Error code parameter not valid.

Note: The Environment Variable APIs provide the *getenv()* (Get Value of Environment Variable) function necessary to retrieve environment variables in ILE/C. Therefore, programs written in ILE/C do not need to use the *QtmhGetEnv()* API. This API, for ILE/C, is more difficult to use (and is slower) than the *getenv()* API on which it is based.

Put Environment Variable (QtmhPutEnv) API

Parameters

Required Parameter Group:

1	Environment string	Input	Char(*)
2	Length of environment string	Input	Binary(4)
3	Error Code	I/O	Char(*)

The *QtmhPutEnv* API allows you to set or create a job-level environment variable. This is useful for communication between programs running in the same job, such as your program and the Net.Data® language environment DTW_SYSTEM.

Required parameter group

Environment string
INPUT:CHAR(*)

The input string of the form: ″envVar=value″. Where ″envVar″ is the name of the new or existing environment variable, and ″value″ is the value you wish to set the environment variable. Note that they are both case sensitive. The server expects this value to be in the CCSID of the job.

Length of environment string
INPUT:BINARY(4)

The input variable that contains the length of the environment string parameter. For example, the length of the environment string ″envVar=value″ is twelve.

Error Code
I/O:CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Error messages

CPF24B4 E
Severe Error while addressing parameter list.

CPF3021 E
The value specified for the argument is not correct.

CPF3C17 E
Error occurred with input data parameter.

CPF3CF1 E
Error code parameter not valid.

CPF3408 E
The address used for an argument is not correct.

CPF3460 E
Storage allocation request failed.

CPF3474 E
Unknown system state.

CPF3484 E
A damaged object was encountered.

Note: The Environment Variable APIs provide the *putenv()* (Put Value in Environment Variable) function necessary to set (or create and set) an environment variable. Therefore, programs written in ILE/C do not need to use the *QtmhPutEnv()* API. This API, for ILE/C, is more difficult to use (and is slower) than the *putenv()* API on which it is based.

Read from Stdin (QtmhRdStin) API

Parameters
Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Length of response available	Output	Binary(4)
4	Error Code	I/O	Char(*)

The *QtmhRdStin* API allows CGI programs that are written in languages other than C to read from stdin. CGI programs read from stdin when the request from the browser indicates the method that is POST. This API reads what the server has generated as input for the CGI program.

Important!
CGI input data is only available from standard input when the client request is submitted with method POST. There are no standard input data when the method is GET or HEAD. In addition, the Content_Length environment variable is set only when the Request_Method is POST.

The program reads all of the data in a single request. This is because the API treats each request as a request for data starting at its beginning. The API handles each request as if it was the only request.

The length of the data returned by *QtmhRdStin* includes all the data from stdin. This includes line-formatting characters that are normally a part of the POST data as defined by the CGI specification.

Note that the format of this data is different depending on the CGI input mode being used. For %%MIXED%% mode, the data will have American National Standard Code for Information Interchange (ASCII) hexadecimal encoded characters. For %%EBCDIC%% mode, all data including hexadecimal will be in the CCSID of the job. The server performs no conversion for %%BINARY%% mode.

Required parameter group

Receiver variable
OUTPUT:CHAR(*)

The output variable that contains the data read from stdin. In CGI input mode %%MIXED%%, this data is in the CCSID of the job except that the encoded characters "%xx" are still represented by the ASCII 819 octet. In %%EBCDIC%% mode, this data is in the CCSID of the job, including the escape sequences. In %%BINARY%% mode, the data is in the code page sent by the browser.

Length of receiver variable
INPUT:BINARY(4)

The input variable containing the number of bytes that are to be read from stdin.

Length or response available
OUTPUT:BINARY(4)

The output variable containing the length of the data read from stdin. If there is no data available from stdin, this variable will be set to zero.

Error Code
I/O:Char(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Error Messages

CPF24B4 E
Severe Error while addressing parameter list.

CPF3C17 E
Error occurred with input data parameter.

CPF3C19 E
Error occurred with receiver variable specified.

CPF3CF1 E
Error code parameter not valid.

Write to Stdout (QtmhWrStout) API

Parameters

Required Parameter Group:

1	Data variable	Input	Char(*)
2	Length of data variable	Input	Binary(4)
3	Error Code	I/O	Char(*)

The *QtmhWrStout* API provides the ability for CGI programs that are written in languages other than C to write to stdout.

Required parameter group

Data variable
Input:CHAR(*)

The input variable containing the data to write to stdout.

Length of data variable
INPUT:BINARY(4)

The input variable contains the length of the data written to stdout. The length of the data must be larger than 0.

Error Code
I/O:CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Error messages

CPF24B4 E
Severe Error while addressing parameter list.

CPF3C17 E
Error occurred with input data parameter.

CPF3CF1 E
Error code parameter not valid.

Note: CGI programs written in the C language do not require a special API to write data to stdout. The following example shows how a CGI program might write to stdout:

```
fwrite(buffer,1,sizeof(buffer),stdout);
```

CGI programs are expected to produce data in the stdout that is formatted according to the CGI interface specification. The *QtmhWrStout* API provides no line formatting; the user of the API must perform prescribed formatting which includes the requirement for text line characters (such as new line). Errors are not indicated for data that is not formatted per CGI requirements.

Convert to DB (QtmhCvtDB) API

Parameters

Required Parameter Group:

1	Qualified database file name	Input	Char(20)
2	Input string	Input	Char(*)
3	Length of input string	Input	Binary(4)
4	Response variable	Output	Char(*)
5	Length of response variable	Input	Binary(4)
6	Length of response available	Output	Binary(4)
7	Response code	Output	Binary(4)
8	Error Code	I/O	Char(*)

The *QtmhCvtDB* API provides an interface for CGI programs to parse CGI input, defined as a series of keywords and their values, into a buffer which is formatted according to a DDS file specification. CGI input data, which comes to the CGI program as character data, will be converted by the *QtmhCvtDB* API to the data type defined for the keyword by the corresponding field name in the input DDS file. AS/400 language statements, such as the ILE C `#pragma mapinc` statement, provide the ability to map the returned structure with field names defined in the DDS file. See the appropriate language user's guide for details.

Note that the *QtmhCvtDB* API is not allowed in CGI mode %%BINARY%%.

The following DDS field types are handled:

A Alphanumeric (see note 1)

P Packed Decimal (see note 2)

S Zoned Decimal

F Floating Point

T Time

L Date

Z Timestamp

B Binary (see note 3)

O DBCS

These DDS field types are not handled:

H Hexadecimal (see note 4)

G Graphic (see note 5)

J DBCS (see note 5)

E DBCS (see note 5)

Notes:

1. The VARLEN keyword is not supported.
2. When using a packed decimal field, the `#pragma mapinc()` must use _P the option, to create a packed structure.

3. Input to Binary fields is converted to integer. The DDS file specification must declare zero decimal positions (for example, “xB 0”, where x is 1-9).
4. ILE C converts hex DDS field data to character fields. Since the input stream to *QtmhCvtDB()* is a text string, the “hex” data would be converted from text to character fields. Therefore, using the A (Alphanumeric) field type to obtain the same conversion.

Required parameter group

Qualified database file name
Input:CHAR(20)

The input variable containing the name of the database file defining field names and data types for the keywords anticipated in the input to the CGI program. Typically, the database file is generated using DDS to define the fields corresponding to the keywords anticipated in the CGI inputs. The first 10 characters contain the database file name, and the second 10 characters contain the library name.

Input string
INPUT:CHAR(*)

The input variable containing the string of CGI input parameters to be parsed. When the environment variable *REQUEST_METHOD* indicates that the method is GET, characters up to the first ? are ignored. The string must meet the format requirements for CGI input keyword strings.

Length of input string
INPUT:BINARY(4)

The input variable containing the length of the character string that contains the CGI input parameters to be parsed. The length of the string must be greater than 0.

Response variable
OUTPUT:CHAR(*)

The output variable which is to contain the structure mapped according to the database file describing the input parameters anticipated by the CGI program.

Length of response available
INPUT:BINARY(4)

The input variable containing the total length of the buffer into which the CGI input parameters will be parsed.

Length of response
OUTPUT:BINARY(4)

The output variable that contains the length of the response. If the response variable is too small to contain the entire response, this parameter will be set to the size that is required to contain the entire response.

Response code
OUTPUT:BINARY(4)

A code that indicates the status of the request.

0 All keywords have been translated according the database file.

-1 The database file contains definitions for structure fields for which the CGI input has no corresponding keyword.

-2 The CGI input contains one or more keywords for which the database file contains no corresponding field.

-3 A combination of the condition for response codes -1 and -2 has been detected.

-4 An error occurred while converting the CGI input string to the DDS defined data types. The data may or may not be usable.

-5 This API is not valid when a program is not called by the IBM HTTP Server. No data parsing is done.

-6 This API is not valid when operating in %%BINARY%% mode. No data parsing is done.

Error Code
I/O CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Error messages

CPF24B4 E
Severe Error while addressing parameter list.

CPF3C17 E
Error occurred with input data parameter.

CPF3C19 E
Error occurred with receiver variable specified.

CPF3CF1 E
Error code parameter not valid.

CPF9810 E
Library &1 not found.

CPF9812 E
File &1 in library &2 not found.

CPF9822 E
Not authorized to file &1 in library &2

Parse QUERY_STRING Environment Variable or Post stdin data (QzhbCgiParse) API

You can use the *QzhbCgiParse* API to parse the QUERY_STRING environment variable, in the case of the GET method, or standard input, in the case of POST method, for CGI scripts. If the QUERY_STRING environment variable is not set, the QzhbCgiParse API reads the CONTENT_LENGTH characters from its input. All return output is written to its standard output.

You can only call QzhbCgiParse once for the POST method. To use this API with the POST method, you would first want to read all of stdin and assign it to the QUERY_STRING environment variable. You would then change the environment variable REQUEST_METHOD to GET.

Parameters

Required Parameter Group:

1	Command string	Input	Char(*)
2	Output format	Input	Char(8)
3	Target Buffer	Output	Char(*)
4	Length of Target Buffer	Input	Binary(4)
5	Length of response	Output	Binary(4)
6	Error Code	I/O	CHAR(*)

Required parameter group

Command string
Input:CHAR(20)

The command string is a null ended string for flags and modifiers. At least one space must separate each flag. There is a one-character equivalent for each flag. The following flags are supported:

-a[gain] *continuation-handle*
The *continuation-handle* is the value returned to the caller in the target buffer when only partial information is returned. This flag is not valid on the first call to this API. It is used to retrieve the next set of information that would have been returned on a previous call if there had been enough space in the target buffer. All other flags must be the same as the previous call. Incomplete or inaccurate information may result if all other flags are not the same.

Note: This flag can only be used for the CGII0200 format.

-k[eywords]
Parses QUERY-STRING for keywords. Keywords are decoded and written to the target buffer, one per line.

-f[orm]
Parses QUERY_STRING as form request. The field names will be set as environment variables with the prefix FORM_. Field values are the contents of the variables.

-v[alue] *field-name*
Parses QUERY_STRING as form request. Returns only the value of *field-name* in the target buffer.

-r[ead]
Reads CONTENT_LENGTH characters from standard input and writes them to the target buffer.

-i[nit]
If QUERY_STRING is not set, reads the value of standard input and returns a string that can be used to set QUERY_STRING.

-s[ep] *separator*
Specifies the string that is used to separate multiple values. If you are using the **-value** flag, the default separation is newline. If you are using the **-form** flag, the default separator is a comma (,).

-p[refix] *prefix*
Used with **-POST** and **-form** to specify the prefix to use when creating environment variable names. The default is ″FORM_″.

-c[ount]
Used with **-keywords**, **-form**, and **-value**, returns a count of items in the target buffer that is related to these flags:

-keywords
Returns the number of keywords.

-form
Returns the number of unique fields (multiple values are counted as one)

-value *field-name*
Returns the number of values for *field-name*. If there is no field that is named *field-name*, the output is 0.

-number
Used with **-keywords**, **-form**, and **-value**. Returns the specified occurrence in the target buffer related to the following flags:

-keywords
Returns the *n*'th keyword. For example, **-2 -keywords** writes the second keyword.

-form
Returns all the values of the *n*'th field.

-value *field-name*
Returns the *n*'th of the multiple values of field *field-name*.

-POST
Information from standard input is directly decoded and parsed into values that can be used to set environment variables. This flag is the equivalent to consecutive use of the **-init** and **-form** options.

-F[sccsid] *FileCCSID*
The *FileCCSID* is the name of the file system CCSID used in CCSID conversion when processing the CGI input data. The CGI program wants the data to be returned in this CCSID. It only applies when the server is using %%BINARY%% CGI conversion mode. When an unknown CCSID is set, the current value of the CGI_EBCDIC_CCSID environment variable is used.

-N[etccsid] *NetCCSID*
The *NetCCSID* is the network CCSID used in CCSID conversion when processing the CGI input data. This is the CCSID that the data is presumed to be in at this time (as assumed or as set in a charset tag). It only applies when the server is using %%BINARY%% CGI Input mode. When an unknown CCSID is set, the current value of the CGI_ASCII_CCSID environment variable is used.

Output format
INPUT:CHAR(*)

The format of the data to be returned in the target buffer. You must use one of the following format names:

- **CGII0100** This format is the free-form format returned to standard output on other platforms.
- **CGII0200** CGI form variable format. This format only applies to the **-form** and **-POST** option.

Target Buffer
OUTPUT:CHAR(*)

This is output buffer that contains the information requested by the command string (if any).

Length of Target Buffer
INPUT:BINARY(4)

The length of the target buffer provided to receive the API output.

Length of Response
OUTPUT:BINARY(4)

The actual length of the information returned in the target buffer.

Error Code
I/O:CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

CGII0200 Format

Offset Decimal	Offset Hexadecimal	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(20)	Continuation handle
28	1C	BINARY(4)	Offset to first variable entry
32	20	BINARY(4)	Number of variable entries returned
36	24	CHAR(*)	Reserved
		BINARY(4)	Length of variable entry (See note)
		BINARY(4)	Length of variable name (See note)
		CHAR(*)	Variable name (See note)
		BINARY(4)	Length of variable value (See note)
		CHAR(*)	Variable value (See note)
		CHAR(*)	Reserved (See note)
Note: These fields contain variable entry information and are repeated for each variable entry returned.			

Field descriptions

Bytes returned The number of bytes of data returned.

Bytes available The number of bytes of data available to be returned. All available data is returned if enough space is available.

Continuation handle The handle that is returned when more data is available to return, but the target buffer is not large enough. The handle indicates the point in the repository that the retrieval stopped. If the handle is used on the next call to the API (using the **-again** flag), the API returns more data starting at the point that the handle indicates. This field is set to blanks when all information is returned.

Offset to first variable entry The offset to the first variable entry returned. The offset is from the beginning of the structure. If no entries are returned, the offset is set to zero.

Number of variable entries returned The number of variable entries returned. If the target buffer is not large enough to hold the information, this number contains only the number of variables actually returned.

Reserved This field is ignored.

Length of variable entry The length of this variable entry. This value is used in determining the offset to the next variable entry. Note that this value is always set to a multiple of four.

Length of variable name The length of the variable name for this entry.

Variable name A field name as found in the form data. If the server is using %%EBCDIC%% or %%MIXED%% CGI mode, this value is in the CCSID of the job. If the server is using %%BINARY%% CGI mode, this value is in the codepage as sent from the browser unless **-fsccsid** is specified on the API invocation. If **-fsccsid** is specified, the value is in that CCSID.

Length of variable value The length of the variable value for this entry.

Variable value A field name as found in the form data. If the server is using %%EBCDIC%% or %%MIXED%% CGI mode, this value is in the CCSID of the job. If the server is using %%BINARY%% CGI mode, this value is in the codepage as sent from the browser unless **-fsccsid** is specified on the API invocation. If **-fsccsid** is specified, the value is in that CCSID.

Error messages

CPF24B4 E
Severe Error while addressing parameter list.

CPF3C17 E
Error occurred with input data parameter.

CPF3C19 E
Error occurred with receiver variable specified.

CPF3CF1 E
Error code parameter not valid.

Note: For further information on errors, the joblog for the CGI job may contain CPF9898 messages (with all English text) describing the error in more detail.

Produce Full HTTP Response (QzhbCgiUtils) API

Use the *QzhbCgiUtils* API to produce a full HTTP 1.0/1.1 response for non-parsed header CGI programs. This API provides functionality similar to the cgiutils command used by other IBM HTTP Server platforms.

Parameters

Required Parameter Group:

1	Command string	Input	Char(*)
2	Error code	I/O	Char(*)

Command string
INPUT:CHAR(*)

The command string is a null ended string of flags and modifiers. Each flag must be separated by at least one space. The following flags are supported:

-nodate
Does not return the Date: header to the browser.

-noel
Does not return a blank line after headers. This is useful if you want other MIME headers after the initial header lines.

-status *nnn*
Returns full HTTP response with status code *nnn*, instead of only a set of HTTP headers. Do not use this flag if you only want the Expires: header.

-reason *explanation*
Specifies the reason line for the HTTP response. You can only use this flag with the **-status** flag. If the explanation text contains more than one word, you must enclose it in parentheses.

-ct [*type/subtype***]**
Specifies MIME Content-Type header to return to the browser. If you omit the *type/subtype*, the MIME content type is set to the default text/plan.

-charset *character-set*
Used with the **-ct** flag to specify the charset tag associated with the text Content-Types.

-ce *encoding*
Specifies MIME Content-Encoding header to return to the browser.

-cl *language-code*
Specifies MIME Content-Language header to return to the browser.

-length *nnn*
Specifies MIME Content-Length header to return to the browser.

-expires *Time-Spec*
Specifies MIME Expires header to return to the browser. This flag specifies the time to live in any combination of years, months, days, hours, minutes, and seconds. The time must be enclosed in parentheses. For example:

```
-expires (2 days 12 hours)
```

-expires now
Produces an Expires: header that matches the Date: header to return to the browser.

-uri *URI*
Specifies the Universal Resource Identifier (URI) for the returned document. URI can be considered the same as URL.

-extra *xxx: yyy*
Specifies an extra header that cannot otherwise be specified.

Error Code
I/O:CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Error messages

CPF24B4 E
Severe Error while addressing parameter list.

CPF3C17 E
Error occurred with input data parameter.

CPF3CF1 E
Error code parameter not valid.

Configuration APIs

The configuration APIs are in *SRVPGM QZHBCONF in library QHTTPSVR. ILE C programs must include header file QHTTPSVR/H(QZHBCONF).

While each individual API lists its own authorities, the following authorities are needed to run all configuration APIs:

- *OBJOPR, *READ, *ADD, and *EXECUTE to the QUSRSYS library
- *READ, *ADD, *DELETE, *EXECUTE, *OBJOPR, *OBJEXIST, and either *OBJMGT or *OBJALTER to the QUSERSYS/QATMHTTPC file
- *READ, *ADD, *DELETE, *EXECUTE, *OBJOPR, *OBJEXIST, and either *OBJMGT or *OBJALTER to the QUSERSYS/QATMHTTPA file

Note: The QUSERSYS/QATMHTTPA file is the administration (ADMIN) server configuration file.

Convert URL to Path (QzhbCvtURLtoPath) API

Required Parameter Group:			
1	Name of Configuration	Input	Char(10)
2	The URL	Input	Char(*)
3	Length of the URL	Input	Binary(4)
4	Path to physical resource	Output	Char(*)
5	Length of path available	Input	Binary(4)
6	Actual length of path	Output	Binary(4)
7	PATH_TRANSLATED	Output	Char(*)
8	Length of PATH_TRANSLATED available	Input	Binary(4)
9	Actual length of PATH_TRANSLATED	Output	Binary(4)
10	QUERY_STRING	Output	Char(*)
11	Length of QUERY_STRING available	Input	Binary(4)
12	Actual length of QUERY_STRING	Output	Binary(4)
13	Error Code	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbCvtURLtoPath* to convert a URL into the physical resource the webserver serves as a result of a request of this URL. All character input and output data will be in the CCSID of the job.

Authorities and locks

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR and **READ authority to the QUSRSYS/QATMHTTPC file

Required parameter group

Name of Configuration
INPUT; CHAR(10)

The name of the configuration from where to retrieve the information.

The URL
INPUT; CHAR(*)

The URL to convert into a physical resource.

Length of the URL
INPUT; BINARY(4)

The length of the URL.

Path to physical resource
OUTPUT; CHAR(*)

The fully qualified path to the physical resource the web server would serve as a result of a request of this URL.

Length of the path available
INPUT; BINARY(4)

The length of the space provided to receive the path to a physical resource.

Actual length of path
OUTPUT; BINARY(4)

The actual path to the physical resource. When the API is unable to determine a physical resource to convert to, this server sets this value to zero. When the size required for the path is larger than the length of the space provided, the actual space required for the path is returned.

PATH_TRANSLATED
OUTPUT; CHAR(*)

The value of PATH_TRANSLATED.

Length of PATH_TRANSLATED available
INPUT; BINARY(4)

The length of the space provided to receive PATH_TRANSLATED.

Actual length of PATH_TRANSLATED
INPUT; BINARY(4)

The actual length of the PATH_TRANSLATED. When API is unable to determine PATH_TRANSLATED, this value will be set to zero. When the size required for PATH_TRANSLATED is larger than the length of the space provided, the actual space required is returned.

QUERY_STRING
OUTPUT; CHAR(*)

The value of QUERY_STRING.

Length of QUERY_STRING available
INPUT; BINARY(4)

The length of the space provided to receive QUERY_STRING.

Actual length of QUERY_STRING
OUTPUT; BINARY(4)

The actual length of the QUERY_STRING. When the API is unable to determine QUERY_STRING, this value will be set to zero. When the size required for QUERY_STRING is larger than the length of the space provided, the actual space required is returned.

Error Code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Error messages

CPF3C17 E
Error occurred with input data parameter.

CPF3CF1 E
Error code parameter not valid.

HTPA104 E
Server configuration not found or is unreadable.

Retrieve Directive (QzhbRetrieveDirective) API

Required Parameter Group:			
1	Name of Configuration	Input	Char(10)
2	Name of the directive	Input	Char(*)
3	Length of the directive name	Input	Binary (4)
4	Number of values returned	Output	Binary (4)
5	Format name	Input	Char(8)
6	Buffer containing length/value pairs	Output	Char(*)
7	Length of space available	Input	Binary (4)
8	Actual length of total values returned	Output	Binary (4)
9	Error code	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbRetrieveDirective* to retrieve the current value of a configuration directive. Some directives can have more than one value. If it does, a list of values is returned in the order found in the configuration file. All character input and output data will be in the CCSID of the job.

Note: The use of this API is discouraged. Support for other more comprehensive configuration APIs (that are descibed in this section) have been provided. This API is being provided for compatibility.

Authorities and locks

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR and **READ authority to the QUSRSYS/QATMHTTPC file

Required parameter group

Name of configuration
INPUT; CHAR(10)

The name of the configuration from which to retrieve the information.

Name of the directive
INPUT; CHAR(*)

The name of the directive to retrieve.

Length of the directive name
INPUT; BINARY(4)

The length of the directive name.

Number of values returned
OUTPUT; BINARY(4)

The number of values returned in the output buffer. This value will be zero if the server finds no matching directives or if there was not enough space available for all the values.

Format name
INPUT; CHAR(8)

The format of the data returned.The possible format names follow:

```
RTVD0100 Retrieve length/value pairs.
```

Buffer containing values
OUTPUT; CHAR(*)

The buffer containing the output.

Length of space available
INPUT; BINARY(4)

The length of the space provided to receive the directive values.

Actual length of total values returned
OUTPUT; BINARY(4)

The actual length of the total values returned. When the API is unable to find a matching directive, this value will be set to zero. When the size required for the total value is larger than the length of the space provided, the actual space required for the total number of values is returned.

Error Code
I/O;CHAR(*)

The structure in which to return error information. For the format of the structure and for details on how to process API errors, see the programming topic in the AS/400 Information Center.

Format of Output Data
The buffer will contain data in one of the following formats:

RTVD0100 Format: The server uses the RTVD0100 format to retrieve a list of length and value pairs for the directive specified.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Displacement to next entry
4	4	CHAR(*)	Value

Error messages

CPF3C17 E
Error occurred with input data parameter

CPF3CF1 E
Error code parameter not valid.

HTPA104 E
Server configuration not found or is unreadable.

Retreive a list of all Configuration Names (QzhbGetConfigNames) API

Required Parameter Group:		
1 buf	Output	Char(*)
2 buf_size	Input	Binary(4)
3 buf_actlen	Output	Binary(4)
4 count	Output	Binary(4)
5 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbGetConfigNames* API to retrieve a list of all server configuration names defined on your AS/400.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR and *READ authority to the QUSRSYS/QATMHTTPC file

Required parameter group

buf
OUTPUT:CHAR(*)

The buffer where the configuration names are placed.

buf_size
INPUT:BINARY(4)

The size of the buffer in bytes.

buf_actlen
OUTPUT:BINARY(4)

The length of all configuration names. Any data beyond the size specified in buf_size parameter is truncated.

count
OUTPUT:BINARY(4)

The total number of server configuration names, whether or not they fit in the buffer.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

CPF9802 E
Not authorized to object &2 &3.

Create a Configuration (QzhbCreateConfig) API

Required Parameter Group:			
	1 name	Input	Char(10)
	2 basedname	Input	Char(10)
	3 basedfile	Input	Char(8)
	4 basedf_len	Input	Binary(4)
	5 errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbCreateConfig* API to create a new configuration. You can create a new configuration based on an existing configuration by passing in an existing configuration name. You can also create an empty configuration file or a configuration file that is based on a text file.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE and *ADD authority to the QUSRSYS library
- *OBJOPR ,*ADD, *DLT, and either *OBJMGT or *OBJALTER authority to the QUSRSYS/QATMHTTPC file

Required parameter group

name
INPUT:CHAR(10)

The configuration file name. The name can be up to 10 characters long (padded with blanks).

basedname
INPUT:CHAR(10)

The name of an existing configuration file used to create a new configuration file. The name can be up to 10 characters long (padded with blanks). To create an empty configuration, pass a NULL (omit) for this parameter or pass all blanks.

basedfile
INPUT:CHAR(8)

The path to the text file used to create the new configuration. This parameter is omissable.

basedf_len
INPUT:BINARY(4)

The length of the basedfile file. A length of 0 means that no basedfile file is passed. If this parameter is greater than 0, basedname cannot also be passed.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPFB602 E
Cannot open file.

HTPA001 E
Input parameter &1 not valid.

HTPA104 E
Server configuration &1 not found or is unreadable.

HTPA105 E
Unable to update server configuration &1.

CPF9802 E
Not authorized to object &2 &3.

Delete a Configuration (QzhbDeleteConfig) API

Required Parameter Group:		
1 name	Input	Char(10)
2 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbDeleteConfig* API to delete a configuration file.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR, *OBJEXIST, *DLT, and either *OBJMGT or *OBJALTER authority to the QUSRSYS/QATMHTTPC file

Required parameter group

name
INPUT:CHAR(10)

The configuration file name you want to delete. The name can be up to 10 characters long (padded with blanks).

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA104 E
Server configuration &1 not found or is unreadable.

HTPA105 E
Unable to update server configuration &1.

CPF9802 E
Not authorized to object &2 &3.

Read a Configuration File into Memory (QzhbOpenConfig) API

Required Parameter Group:		
1 name	Input	Char(10)
2 writelock	Input	Binary(4)
3 cfg	Output	Binary(4)
4 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbOpenConfig* API to read a configuration file into memory. A handle to the memory copy of the file is returned, and is used in subsequent API calls to manipulate directives within the file. When the copy of the file is no longer required, the QzhbCloseConfig API is used to free it and optionally write the altered contents out.

Authorities and locks

To invoke this API with a writelock value of 0, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR and *READ authority to the QUSRSYS/QATMHTTPC file

To invoke this API with a writelock value of 1, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR, *OBJMGR, *ADD, and *DLT authority to the QUSRSYS/QATMHTTPC file

Required parameter group

name
INPUT:CHAR(10)

The name of the configuration file you want to read into memory. The name can be up to 10 characters long (padded with blanks).

writelock
INPUT:BINARY(4)

The value 0 (false) or 1 (true). If the value is 1, an exclusive read object lock is obtained on this member of the QUSRSYS/QATMHTTPC file. No other user can update the configuration while the lock is in place. The lock is released when the QzhbCloseConfig API is called. If the value is 0, no lock is placed on the member.

Note: You must specify a writelock of 1, and successfully obtain the object lock, in order to later specify a write argument of 1 on the QzhbCloseConfig API. If you do not have this lock, the QzhbCloseConfig API will not write the contents of the configuration file.

cfg
OUTPUT:BINARY(4)

The handle returned to the loaded configuration file.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA104 E
Server configuration &1 not found or is unreadable.

HTPA105 E
Unable to update server configuration &1.

CPF9802 E
Not authorized to object &2 &3.

Free a Configuration File from Memory (QzhbCloseConfig) API

Required Parameter Group:			
1	cfg	Input	Binary(4)
2	write	Input	Binary(4)
3	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbCloseConfig* API to free a configuration file in memory. Optionally, the data in memory can first be written to the configuration file where it was read from by the QzhbOpenConfig API.

Authorities and locks
None.

Required parameter group

cfg
INPUT:BINARY(4)

The configuration file handle, returned by a call to API QzhbOpenConfig.

write
INPUT:BINARY(4)

When 1 is specified in the write parameter, the directives are written to the configuration file before being freed from memory. If a write fails, the memory is not freed, the handle is still valid, and error information is returned.

When 0 is specified, the dirrectives are not written, but the object lock is released if it was obtained at QzhbOpenConfig time.

Note: In order to specify a write of 1, you must have previously specified a writelock of 1 on the QzhbOpenConfig API.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA105 E
Unable to update server configuration &1.

HTPA106 E
Input configuration handle not valid.

Search for a Main Directive (QzhbFindDirective) API

Required Parameter Group:			
1	cfg	Input	Binary(4)
2	value	Input	Char(*)
3	value_len	Input	Binary(4)
4	startdir	Input	Binary(4)
5	num	Input	Binary(4)
6	case_sens	Input	Binary(4)
7	dir	Output	Binary(4)
8	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbFindDirective* API to find a main directive in a configuration file previously opened by a call to the QzhbOpenConfig API. If a directive is found, a handle to the directive is returned and can be used on subsequent calls to other APIs.

Authorities and locks
None.

Required parameter group

cfg
INPUT:BINARY(4)

The handle returned by a call to the QzhbOpenConfig API.

value
INPUT:BINARY(4)

The character string for matching to a directive. Only as many tokens (words delimited by a space) as are provided are matched. Any extra tokens either on the value string or the directive being considered for a match will not be compared. For example a value string of `Port 1234 junk` will match a directive of `Port 1234`. To match any directive, including comment lines, pass either a NULL pointer or a string with no tokens on it such as a 0 length string.

value_len
INPUT:BINARY(4)

The length of the **value** string.

startdir
INPUT:BINARY(4)

The directive handle that specifies where to begin searching for a match. The directive immediately following this one is the first one searched. If the startdir parameter is passed as a NULL, then searching begins at the beginning of the configuration file. If the startdir parameter is not passed as a NULL (omitted), then the startdir parameter must be the handle to a main directive, and cannot be a subdirective.

num
INPUT:BINARY(4)

The number of the match to be returned. The num parameter must be a number greater than or equal to 0. If the value is 0, then the last matching directive is returned. If the value is 1, the first match is returned. If the value is 2, the second match is returned, and so on.

case_sens
INPUT:BINARY(4)

The value of 0 (false) or a value of 1 (true), indicating whether matching of tokens in the search string should be case sensitive. In most cases, except where certain case-sensitive file paths are being considered, this parameter should be 0 (false). Note that the searches for the actual directive name, which is the first token on the line, is never case-sensitive.

dir
OUTPUT:BINARY(4)

The handle to the matched directive. If no directive is found, error HTPA110 is returned.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA106 E
Input configuration handle not valid.

HTPA107 E
Input directive handle in parameter &1 not valid.

HTPA108 E
Input directive handle in parameter &1 not a main directive.

HTPA110 E
No matching directive found.

Search for a Subdirective under Main Directive (QzhbFindSubdirective) API

Required Parameter Group:			
1	cfg	Input	Binary(4)
2	maindir	Input	Binary(4)
3	value	Input	Char(*)
4	value_len	Input	Binary(4)
5	startdir	Input	Binary(4)
6	num	Input	Binary(4)
7	case_sens	Input	Binary(4)
8	dir	Output	Binary(4)
9	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbFindSubdirective* API to find a subdirective in a configuration file previously opened by a call to the QzhbOpenConfig API. If a subdirective is found, a handle to the subdirective is returned and can be used on subsequent calls to other APIs.

Authorities and locks
None.

Required parameter group

cfg
INPUT:BINARY(4)

The handle returned by a call to the QzhbOpenConfig API.

maindir
INPUT:BINARY(4)

The handle to a main directive previously returned by a call to the QzhbFindDirective API. If the startdir parameter is not NULL, then the maindir parameter can be passed as NULL since the main directive is implied by startdir. If both the maindir and startdir parameters are passed, then the startdir parameter must specify the handle to a subdirective under maindir.

value
INPUT:CHAR(*)

The character string for matching to a subdirective. Only as many tokens (words delimited by a blank) as are provided are matched. Any extra tokens either on the value string or the subdirective being considered for a match will not be compared. For example a value string of `Port 1234 junk` will match a directive of `Port 1234`. To match any subdirective, including comment lines, pass either a NULL pointer or a string with no tokens on it such as a 0 length string.

value_len
INPUT:BINARY(4)

The length of the **value** string.

startdir
INPUT:BINARY(4)

The subdirective handle that specifies where to begin searching for a match. The subdirective immediately following this one is the first one searched. If the startdir parameter is passed as a NULL (omitted), then searching begins at the beginning of the subdirective list for maindir. If the startdir parameter is not NULL, then the maindir parameter can be passed as NULL since the main directive is implied by startdir. If both the maindir and startdir parameters are passed, then startdir must be the handle to a subdirective under maindir.

num
INPUT:BINARY(4)

The number of the match to be returned. The num parameter must be a number greater than or equal to 0. If the value is 0, then the last matching subdirective is returned. If the value is 1, the first match is returned. If the value is 2, the second match is returned, and so on.

case_sens
INPUT:BINARY(4)

The value of 0 (false) or a value of 1 (true), indicating whether matching of tokens in the search string should be case sensitive. In most cases, except where certain case-sensitive file paths are being considered, this parameter should be 0 (false). Note that the searches for the actual subdirective name, which is the first token on the line, is never case-sensitive.

dir
OUTPUT:BINARY(4)

The handle to the matched subdirective, if found.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA106 E
Input configuration handle not valid.

HTPA107 E
Input directive handle in parameter &1 not valid.

HTPA108 E
Input directive handle in parameter &1 not a main directive.

HTPA109 E
Input directive handle in parameter &1 not a subdirective.

HTPA110 E
No matching directive found.

Return Details of a Main Directive or Subdirective (QzhbGetDirectiveDetail) API

Required Parameter Group:		
1 cfg	Input	Binary(4)
2 dir	Input	Binary(4)
3 buf	Output	Char(*)
4 buf_size	Input	Binary(4)
5 buf_actlen	Output	Binary(4)
6 hassubdirs	Output	Binary(4)
7 issubdir	Output	Binary(4)
8 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbGetDirectiveDetail* API to extract detail information about a main directive or subdirective.

Authorities and locks
None.

Required parameter group
cfg
INPUT:BINARY(4)

The handle returned to the configuration file by a call to API QzhbOpenConfig.

dir
INPUT:BINARY(4)

The handle to a main directive or subdirective, as returned by the QzhbFindDirective or QzhbFindSubdirective APIs.

buf
OUTPUT:CHAR(*)

The buffer where the directive string is placed.

buf_size
INPUT:BINARY(4)

The size of the buffer in bytes.

buf_actlen
OUTPUT:BINARY(4)

The actual length of the directive string. Any data beyond the size specified in the buf_size parameter is truncated.

hassubdirs
OUTPUT:BINARY(4)

The value is set to 1 (true) when dir is a main directive and there are subdirectives under it. If dir is not a main directive, the value is set to 0 (false).

issubdir
OUTPUT:BINARY(4)

The value is set to 1 (true) when dir is a subdirective. The value is set to 0 (false) when dir is a main directive.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA106 E
Input configuration handle not valid.

HTPA107 E
Input directive handle in parameter &1 not valid.

Add a Main Directive or Subdirective (QzhbAddDirective) API

Required Parameter Group:			
1	cfg	Input	Binary(4)
2	value	Input	Char(*)
3	value_len	Input	Binary(4)
4	position	Input	Binary(4)
5	reldir	Input	Binary(4)
6	newdir	Output	Binary(4)
7	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbAddDirective* API to add a new main directive or subdirective to a configuration file located in memory.

Authorities and locks
None.

Required parameter group

cfg
INPUT:BINARY(4)

The handle returned by a call to API QzhbOpenConfig.

value
INPUT:CHAR(*)

The character string of the new directive. This string is not validated in any way to ensure that it is a valid directive.

value_len
INPUT:BINARY(4)

The length of the value string. The length must be greater than or equal to 1.

position
INPUT:BINARY(4)

The number indicating the insertion position for the new directive. See Table 3 for more information.

reldir
INPUT:BINARY(4)

The handle to a main directive or subdirective, or a NULL (omitted). See Table 3 for more information.

newdir
OUTPUT:BINARY(4)

The handle of the newly added main directive or subdirective.

errcode
I/O:CHAR(*)

The structure in which to return error information.

The position and reldir parameters must be considered together. The combination of these parameters determine whether the directive being added is a main directive or subdirective and where in the configuration file it is to be added. Table 3 shows the behavior for the various combinations of these parameters.

Table 3. Using the reldir and position parameters.

reldir value	Position 0 (Before)	Position 1 (After)	Position 2 (At front)	Position 3 (At end)	Position 4 (Automatic)
NULL (omitted)	Not valid	Not valid	Inserted as a main directive at the beginning of the file.	Inserted as a main directive at the end of the file.	Inserted as a main directive at a location determined by internal rules of directive ordering. Use this mode when you are not sure where to insert.
Main directive	Inserted as a main directive directly proceeding redir.	Inserted as a main directive directly following reldir.	Inserted as a subdirective at the front of this directive's subdirective list.	Inserted as a subdirective at the end of this directive's subdirective list, but proceeding the close brace subdirective "}".	Inserted as a subdirective at a location in this directive's subdirective list as determined by internal rules of subdirective ordering. Use this mode when you are not sure where to insert.

Table 3. Using the reldir and position parameters. (continued)

reldir value	Position 0 (Before)	Position 1 (After)	Position 2 (At front)	Position 3 (At end)	Position 4 (Automatic)
Subdirective	Inserted as a subdirective directly proceeding reldir.	Inserted as a subdirective directly following reldir.	Not valid	Not valid	Not valid

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA106 E
Input configuration handle not valid.

HTPA107 E
Input directive handle in parameter &1 not valid.

HTPA111 E
Combination of insertion position and relative directive not valid.

Remove a Main Directive or Subdirective (QzhbRemoveDirective) API

Required Parameter Group:			
	1 cfg	Input	Binary(4)
	2 dir	Input	Binary(4)
	3 errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbRemoveDirective* API to remove a main directive or subdirective from a configuration file located in memory.

Authorities and locks
None.

Required parameter group

cfg
INPUT:BINARY(4)

The handle returned by a call to API QzhbOpenConfig.

dir
INPUT:BINARY(4)

The handle to the main directive or subdirective to be removed.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA106 E
Input configuration handle not valid.

HTPA107 E
Input directive handle in parameter &1 not valid.

Replace a Main Directive or Subdirective (QzhbReplaceDirective) API

Required Parameter Group:			
1	cfg	Input	Binary(4)
2	dir	Input	Binary(4)
3	value	Input	Char(*)
4	value_len	Input	Binary(4)
5	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbReplaceDirective* API to replace the string value of a main directive or subdirective in a configuration file located in memory.

Authorities and locks
None.

Required parameter group

cfg
INPUT:BINARY(4)

The handle returned by a call to the QzhbOpenConfig API.

dir
INPUT:BINARY(4)

The handle to the main directive or subdirective to be changed.

value
INPUT:CHAR(*)

The character string to replace the directive. This string is not validated in any way to ensure that it is a valid directive.

value_len
INPUT:BINARY(4)

The length of the value string must be greater than or equal to 1.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA106 E
Input configuration handle not valid.

HTPA107 E
Input directive handle in parameter &1 not valid.

Server instance APIs

The server instance APIs are in *SRVPGM QZHBCONF in library QHTTPSVR. ILE C programs must include header file QHTTPSVR/H(QZHBCONF).

While each individual API lists its own authorities, the following authorities are needed to run all server instance APIs:

- *OBJOPR, *READ, *ADD, and *EXECUTE to the QUSRSYS library
- *READ, *ADD, *DELETE, *EXECUTE, *OBJOPR, *OBJEXIST, and either *OBJMGT or *OBJALTER to the QUSERSYS/QATMHINSTC file
- *READ, *ADD, *DELETE, *EXECUTE, *OBJOPR, *OBJEXIST, and either *OBJMGT or *OBJALTER to the QUSERSYS/QATMHINSTA file

Note: The QUSERSYS/QATMINSTA file is the administration (ADMIN) server instance file.

Retrieve a list of all Server Instances (QzhbGetInstanceNames) API

Required Parameter Group:			
1	buf	Output	Void
2	buf_size	Input	Binary(4)
3	format	Input	Char(8)
4	buf_actlen	Output	Binary(4)
5	count	Output	Binary(4)
6	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbGetInstanceNames* API to retrieve a list of all server instance names defined on your AS/400.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR and *READ authority to the QUSRSYS/QATMHINSTC file

Required parameter group

buf
OUTPUT:VOID

The buffer where the instance names are placed. Specify a buffer name up to 10 characters (padded with blanks) as necessary.

buf_size
INPUT:BINARY(4)

The size of the buffer in bytes.

format
INPUT:CHAR(8)

The format of the data returned. The possible format names follow:

`INSN0100`

buf_actlen
OUTPUT:BINARY(4)

The length of all the instance names. Any data beyond the size specified in the buf_size value is truncated by the system.

count
OUTPUT:BINARY(4)

The total number of instance names.

errcode
I/O:CHAR(*)

The structure in which to return error information.

INSN0100 Format

INSN0100 Format:

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Instance name
10	0A	CHAR(2)	Reserved
12	0C	BINARY(4)	Running status

Field descriptions

Instance name
The 10 character name of the server instance, padded with blanks.

Running status
An integer value of 1 if this instance is currently running. An integer value of 0 if this server instance is currently stopped.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

CPF3C21 E
Format name &1 not valid.

HTPA001 E
Input parameter &1 not valid

CPF9802 E
Not authorized to object &2 in &3.

Look up Server Instance Data (QzhbGetInstanceData) API

Required Parameter Group:			
1	name	Input	Char(10)
2	buf	Output	Void
3	buf_size	Input	Binary(4)
4	format	Input	Char(8)
5	buf_actlen	Output	Binary(4)
6	running	Output	Binary(4)
7	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbGetInstanceData* API to get detailed data about a specific server instance. This data includes whether the instance is currently running and all start-up data.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR and *READ authority to the QUSRSYS/QATMHINSTC file

Required parameter group

name
INPUT:CHAR(10)

The server instance name. The name can be up to 10 characters long (padded with blanks).

buf
OUTPUT:VOID

The buffer where the instance names are placed. The buffer is defined based on the format parameter. You may omit this parameter.

buf_size
INPUT:BINARY(4)

The size of the buffer in bytes. A size of 0 can be specified meaning that no data is returned in the buffer, but the running variable is still set to indicate the running status of the instance.

format
INPUT:CHAR(8)

The format in which the data should be returned. The possible format names follow:

```
INSD0100
```

buf_actlen
OUTPUT:BINARY(4)

The number of bytes available for instance data. For the INSD0100 format, the buf_actlen value is 1104 bytes.

running
OUTPUT:BINARY(4)

Indicates if the server instance is running. If the instance is running, the running parameter is set to 1. If the instance is not running, the running parameter is set to 0. The running parameter can be omitted. If this value is omitted (null), the running status is not queried by this API, and no performance penalty is incurred for finding this information. See "Retrieve a list of all Server Instances (QzhbGetInstanceNames) API" on page 56 for another method to query the running status of all instances.

errcode
I/O:CHAR(*)

The structure in which to return error information.

INSD0100 Format

Offset Decimal	Offset Hexadecimal	Type	Field
0	0	CHAR(10)	Configuration
10	0A	CHAR(10)	Autostart
20	14	BINARY(4)	Min threads
24	18	BINARY(4)	Max threads
28	1C	BINARY(4)	CCSID
32	2A	CHAR(10)	Outgoing table name
42	20	CHAR(10)	Outgoing table library
52	3E	CHAR(10)	Incoming table name
62	34	CHAR(10)	Incoming table library
72	48	CHAR(512)	Access log file
584	248	CHAR(512)	Error log file
1096	448	BINARY(4)	Non-secure port
1100	44C	BINARY(4)	Secure port

Field descriptions

Note: In the descriptions below, *GLOBAL indicates that the global server parameter value for this field is used by the instance, and *CFG indicates that the value from the named configuration file is used. All character strings are padded with blanks as necessary, and are NOT null terminated.

Configuration
The 10 character name of the configuration used for this instance.

Autostart
Indicates if the instance starts automatically. It is a 10 character string that contains *NO, *YES, or *GLOBAL.

Min threads
The minimum number of threads to use for this instance. It is an integer from 0 to 999, where 0 means the *CFG value.

Max threads
The maximum number of threads to use for this instance. It is an integer from -1 to 999, where 0 means the *CFG value and -1 means *NOMAX (no maximum).

CCSID
The character set to be used by the instance. It is an integer from 0 to 65533, where 0 means *GLOBAL.

Outgoing table name
The name of the table object to use as the EBCDIC to ASCII conversion table for outgoing data. It is a 10 character name or *GLOBAL.

Outgoing table library
The library containing the EBCDIC to ASCII table. This field is blank if the outgoing table name is *GLOBAL.

Incoming table name
The name of the table object to use as the ASCII to EBCDIC conversion table for incoming data. It is a 10 character name or *GLOBAL.

Incoming table library
The library containing the ASCII to EBCDIC table. This field is blank if the incoming table name is *GLOBAL.

Access log file
The path to the access log file as a 512 character string. This is an IFS type path name in the job CCSID, or *CFG.

Error log file
The path to the access log file as a 512 character string. This is an IFS type path name in the job CCSID, or *CFG

Non-secure port
The TCP port where the server will listen for normal HTTP connections. It is an integer from 0 to 65535, where 0 means *CFG.

Secure port
The TCP port where the server will listen for secure SSL HTTPS connections. It is an integer from 0 to 65535, where 0 means *CFG.

Error messages

CPF3C21 E
Format name &1 not valid.

CPF3CF1 E
Error code parameter not valid.

CPF9802 E
Not authorized to object &2 in &3.

HTPA001 E
Input parameter &1 not valid.

HTPA101 E
Server instance &1 not found or is unreadable.

Change Server Instance Data (QzhbChangeInstanceData) API

Required Parameter Group:		
1 name	Input	Char(10)
2 idata	Input	Void

3	idata_size	Input	Binary(4)
4	format	Input	Char(8)
5	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbChangeInstanceData* API to change the start-up data for a specific server instance. This API provides a structure for input, even when not changing values, to all start-up data values to be set for a server instance. This API is typically used following a call to the QzhbGetInstanceData API, and after one or more fields in the structure have been modified.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR, *OBJMGT, *ADD, and *DLT authority to the QUSRSYS/QATMHINSTC file

Required parameter group

name
INPUT:CHAR(10)

The server instance name. The name can be up to 10 characters long (padded with blanks).

idata
INPUT:VOID

The buffer where the instance data is stored. The contents of the buffer is defined by the format specifed by the format parameter. All fields in the idata parameter must contain valid values.

idata_size
INPUT:BINARY(4)

The size of the idata structure. The minimum size is the length needed for the INSD0100 format, 1104 bytes.

format
INPUT:CHAR(8)

The format of the data returned. The possible format names follow:

```
INSD0100
```

For information about the INSD0100 format, see "INSD0100 Format" on page 59.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

CPF3C21 E
Format name &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA101 E
Server instance &1 not found or is unreadable.

HTPA102 E
Unable to update server instance &1.

HTPA103 E
Value in field &1 of the instance data structure not valid.

Create a Server Instance (QzhbCreateInstance) API

Required Parameter Group:			
1	name	Input	Char(10)
2	idata	Input	Void
3	idata_size	Input	Binary(4)
4	format	Input	Char(8)
5	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbCreateInstance* API to create a new server instance. This API provides a structure for input to all start-up data values to be set for a new server instance. Use this API following a call to the QzhbGetInstanceData API to create an instance based on an existing instance.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE and *ADD authority to the QUSRSYS library
- *OBJOPR, *ADD, *DLT, and either *OBJMGT or *OBJALTER authority to the QUSRSYS/QATMHINSTC file

Required parameter group

name
INPUT:CHAR(10)

The name for the new server instance you want to create. The name can be up to 10 characters long (padded with blanks).

idata
INPUT:VOID

The buffer where the instance data is stored. The contents of the buffer is defined by the format specifed by the format parameter. All fields in the idata parameter must contain valid values.

idata_size
INPUT:BINARY(4)

The size of the idata structure. The minimum size is the length needed for the INSD0100 format, 1104 bytes.

format
INPUT:CHAR(8)

The format of the data returned. The possible format names follow:
`INSD0100`

For information about the INSD0100 format, see "INSD0100 Format" on page 59.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

CPF3C21 E
Format name &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA102 E
Unable to update server instance &1.

HTPA103 E
Value in field &1 of the instance data structure not valid.

CPF9802 E
Not authorized to object &2 &3.

Delete a Server Instance (QzhbDeleteInstance) API

Required Parameter Group:		
1 name	Input	Char(10)
2 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbDeleteInstance* API to delete a server instance.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *EXECUTE authority to the QUSRSYS library
- *OBJOPR, *OBJEXIST, *DLT and either *OBJMGT or *OBJALTER authority to the QUSRSYS/QATMHINSTC file

Required parameter group

name
INPUT:CHAR(10)

The server instance name you want to delete. The name can be up to 10 characters long (padded with blanks).

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid

HTPA101 E
Server instance &1 not found or is unreadable.

HTPA102 E
Unable to update server instance &1.

CPF9802 E
Not authorized to object &2 &3.

Group file APIs

The group file APIs are in *SRVPGM QZHBCONF in library QHTTPSVR. ILE C programs must include header file QHTTPSVR/H(QZHBCONF).

Create a new Group File (QzhbCreateGroupList) API

Required Parameter Group:			
1	path	Input	Binary(4)
2	path_len	Input	Binary(4)
3	grplist	Output	Binary(4)
4	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbCreateGroupList* API to create a new empty group file, and return a handle to that empty in-memory version of the file. Normally this API would be followed by calls to the QzhbAddGroupToList and QzhbAddUserToGroup APIs, followed by the QzhbCloseGroupList API to write group information out.

Upon successful completion of this API, a new group list handle is returned. This is a handle much like the one returned by the QzhbOpenGroupList API against an already existing file, with a writelock argument of 1 (TRUE). After a call to the QzhbCreateGroupList API the new file is left open for write access and the QzhbCloseGroupList API can be invoked with a write argument of 1. For more details about the writelock argument, see “Read a Group File into Memory (QzhbOpenGroupList) API” on page 65.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *X authority to each directory in the path of the specified group file
- *WX authority to the last directory in the path that will contain the group file path

Required parameter group

path
INPUT:BINARY(4)

The path to the group file to be created in the Integrated File System. You can specify an absolute or relative path to the working directory. This path should be in the job CCSID.

path_len
INPUT:BINARY(4)

The length of the path string.

grplist
OUTPUT:BINARY(4)

The variable that receives the integer handle of the newly created empty group list. Subsequent API calls use this handle.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA202 E
Unable to update group file &1.

HTPA208 E
Group file &1 already exists.

Read a Group File into Memory (QzhbOpenGroupList) API

Required Parameter Group:			
1	path	Input	Binary(4)
2	path_len	Input	Binary(4)
3	writelock	Input	Binary(4)
4	grplist	Output	Binary(4)
5	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbOpenGroupList* API to read in an existing group file, and return a handle to an in-memory version of the file. See "Free Group File from Memory (QzhbCloseGroupList) API" on page 67 for information about freeing memory and optionally writing the group list information out.

Authorities and locks

To invoke this API, the user must have the following authorities:

- *X authority to each directory in the path of the specified group file

- *R authority to the group file for a writelock value of 0
- *RW authority to the group file for a writelock value of 1

Required parameter group

path
INPUT:BINARY(4)

The path to the group file to be created in the Integrated File System. You can specify an absolute or relative path to the working directory.

path_len
INPUT:BINARY(4)

The length of the path string.

writelock

If the value is 1, the group file is opened for write access with a lock and kept open. No other user is allowed to update the group file while the lock is in place. The group file is closed and the lock released by invoking the QZHbCloseGroupList API. If the value is 0, then the following are true:

- The group file is opened for read access
- A lock is not placed on the group file
- The group file does not remain open

Note: You must specify a writelock of 1 in order to later specify a write argument of 1 on the QzhbCloseGroupList API. If you do not hold the group file open for write, the QzhbCloseGroupList API will not write the contents of the file.

grplist
OUTPUT:BINARY(4)

The handle of the group list. Subsequent API calls use this handle.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA201 E
Group file &1 not found or is unreadable.

HTPA202 E
Unable to update group file &1.

Free Group File from Memory (QzhbCloseGroupList) API

Required Parameter Group:		
1 grplist	Input	Binary(4)
2 write	Input	Binary(4)
3 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbCloseGroupList* API to free the memory of an in-memory copy of a group file. You can optionally write the in-memory version of the group list back to the group file before the memory is freed.

Authorities and locks

None.

Required parameter group

grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList API or QzhbOpenGroupList API.

write
INPUT:BINARY(4)

The value of 0 (false) or a value of 1 (true), indicating whether or not to write the contents of the in-memory group list back to the group file before freeing it from memory. If you specify 1 for this parameter, and the write fails, the memory is not freed and the grplist handle is still valid.

Note: In order to specify a write value of 1, you must have previously used either the QzhbCreateConfigList API or specified a writelock of 1 on the QzhbOpenGroupList API. If these conditions are not met, the contents of the file are not written.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA202 E
Unable to update group file &1.

HTPA203 E
Input group list handle in parameter &1 not valid.

Retrieve the next Group in the Group List (QzhbGetNextGroup) API

Required Parameter Group:		
1 grplist	Input	Binary(4)
2 prev_grp	Input	Binary(4)
3 grp	Output	Binary(4)
4 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbGetNextGroup* API to retrieve the next group from an in-memory group list.

Authorities and locks

None.

Required parameter group

grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

prev_grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API, that returns the group immediately following this group. A handle of 0 returns the first group in the group list.

grp
OUTPUT:BINARY(4)

The group name handle returned if the next group is found in the list. If no next group exists, then error HTPA206 is returned.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

HTPA206 E
Group file &1 not found in group list.

Locate a named group in a Group List (QzhbFindGroupInList) API

Required Parameter Group:			
	1 grplist	Input	Binary(4)
	2 group	Input	Binary(4)
	3 group_len	Input	Binary(4)
	4 grp	Output	Binary(4)
	5 errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbFindGroupInList* API to search an in-memory group list for a named group.

Authorities and locks

None.

Required parameter group

grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

group
INPUT:CHAR(*)

The group name for which the system will search the list . The group name is case-sensitive. Leading and trailing blanks are included with the name.

group_len
INPUT:BINARY(4)

The length of the group name string. The length must be greater than or equal to 1.

grp
OUTPUT:BINARY(4)

The group name handle returned if the named group was found in the list.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA206 E
Group file &1 not found in group list.

Retrieve the Name of a Group (QzhbGetGroupName) API

Required Parameter Group:			
1	grplist	Input	Binary(4)
2	grp	Input	Binary(4)
3	buf	Output	Char(*)
4	buf_len	Input	Binary(4)
5	buf_actlen	Output	Binary(4)
6	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbGetGroupName* API to retrieve the name of a group using the group handle.

Authorities and locks
None.

Required parameter group
grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API.

buf
OUTPUT:BINARY(4)

The buffer to receive the group name.

buf_len
OUTPUT:BINARY(4)

The size of the buffer.

buf_actlen
OUTPUT:BINARY(4)

The actual length of the group name. If the buf_actlen value is greater than the buf_len value, the data is truncated.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages
CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

Add a new Group to the end of a Group List (QzhbAddGroupToList) API

Required Parameter Group:		
1 grplist	Input	Binary(4)
2 group	Input	Char(*)
3 group_len	Input	Binary(4)
4 grp	Output	Binary(4)
5 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbAddGroupToList* API to add a new group to an in-memory group list.

Authorities and locks
None.

Required parameter group
grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

group
INPUT:CHAR(*)

The group name to add to the list.

group_len
INPUT:BINARY(4)

The length of the group name. The length must be greater than or equal to 1.

grp
OUTPUT:BINARY(4)

The handle of the newly created group, or the handle of an existing group if the named group already exists. Attempting to add a group that already exists is not considered an error by the system.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

Remove a Group from a Group List (QzhbRemoveGroupFromList) API

Required Parameter Group:		
1 grplist	Input	Binary(4)
2 grp	Input	Binary(4)
3 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbRemoveGroupFromList* API to remove a named group, and all the users in that group, from an in-memory group list.

Authorities and locks
None.

Required parameter group

grplist
INPUT:BINARY(4)

The group handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

Retrieve the next User in the Group (QzhbGetNextUser) API

Required Parameter Group:			
1	grplist	Input	Binary(4)
2	grp	Input	Binary(4)
3	prev_usr	Input	Binary(4)
5	usr	Output	Binary(4)
6	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbGetNextUser* API to retrieve the next user from a group.

Authorities and locks

None.

Required parameter group

grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API.

prev_usr
INPUT:BINARY(4)

The user handle for an existing user that returns the user immediately following this user. A handle of 0 returns the first user in the group list.

usr
OUTPUT:BINARY(4)

The handle of the user if a next user is found in the group. If no next user is found, error HTPA207 is returned.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

HTPA205 E
Input user handle in parameter &1 not valid.

HTPA207 E
User &1 not found in group.

Locate a User in a Group (QzhbFindUserInGroup) API

Required Parameter Group:			
1	grplist	Input	Binary(4)
2	grp	Input	Binary(4)
3	user	Input	Char(*)
4	user_len	Input	Binary(4)
5	usr	Output	Binary(4)
6	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbFindUserInGroup* API to search an in-memory group for a specific user.

Authorities and locks
None.

Required parameter group
grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API.

user
INPUT:CHAR(*)

The user name for which the system will search the group . The user name is case-sensitive. Leading and trailing blanks are included with the name.

user_len
INPUT:BINARY(4)

The length of the user string. The length must be greater than or equal to 1.

usr
OUTPUT:BINARY(4)

The handle of the user if it was found in the group.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

HTPA207 E
User &1 not found in group.

Retrieve the Name of a User (QzhbGetUserString) API

Required Parameter Group:			
1	grplist	Input	Binary(4)
2	grp	Input	Binary(4)
3	usr	Input	Binary(4)
4	buf	Output	Char(*)
5	buf_len	Input	Binary(4)
6	buf_actlen	Output	Binary(4)
7	errcode	I/O	Char(*)
Threadsafe: Yes			

Use the *QzhbGetUserString* API to retrieve the name string of a group member given the user handle, as returned by the QzhbGetNextUser, QzhbFindUserInGroup, or QzhbAddUserToGroup API.

Authorities and locks
None.

Required parameter group

grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API.

usr
INPUT:BINARY(4)

The user handle returned from a call to the QzhbGetNextUser, QzhbFindUserInGroup, or QzhbAddUserToGroup API.

buf
OUTPUT:CHAR(*)

The buffer to receive the user string.

buf_len
INPUT:BINARY(4)

The size of the buffer.

buf_actlen
OUTPUT:BINARY(4)

The actual length of the user string. If the buf_actlen value is greater than the buf_len value, the data is truncated by the system.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

HTPA205 E
Input group handle in parameter &1 not valid.

Add a new user to the end of a Group (QzhbAddUserToGroup) API

Required Parameter Group:		
1 grplist	Input	Binary(4)
2 grp	Input	Binary(4)
3 user	Input	Char(*)
4 user_len	Input	Binary(4)
5 usr	Output	Binary(4)
6 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbAddUserToGroup* API to add a new user to an in-memory group.

Authorities and locks
None.

Required parameter group

grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API.

user
INPUT:CHAR(*)

The user name to be added to the group.

user_len
INPUT:BINARY(4)

The length of the user string. The length must be greater than or equal to 1.

usr
OUTPUT:BINARY(4)

The handle of the newly created user, or the handle of an existing user if the named user already exists in the group. Attempting to add a user that already exists is not considered an error by the system.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

CPF3C1D E
Input variable length in parameter &1 not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

Remove a User or Element from a Group (QzhbRemoveUserFromGroup) API

Required Parameter Group:			
1	grplist	Input	Binary(4)
2	grp	Input	Binary(4)
3	usr	Input	Binary(4)

4 errcode	I/O	Char(*)
Threadsafe: Yes		

Use the *QzhbRemoveUserFromGroup* API to remove a user from an in-memory group.

Authorities and locks

None.

Required parameter group

grplist
INPUT:BINARY(4)

The group list handle returned from a call to the QzhbCreateGroupList or QzhbOpenGroupList API.

grp
INPUT:BINARY(4)

The group handle returned from a call to the QzhbGetNextGroup, QzhbFindGroupInList, or QzhbAddGroupToList API.

usr
INPUT:BINARY(4)

The user handle returned from a call to the QzhbGetNextUser, QzhbFindUserInGroup, or QzhbAddUserToGroup API.

errcode
I/O:CHAR(*)

The structure in which to return error information.

Error messages

CPF3CF1 E
Error code parameter not valid.

HTPA001 E
Input parameter &1 not valid.

HTPA203 E
Input group list handle in parameter &1 not valid.

HTPA204 E
Input group handle in parameter &1 not valid.

HTPA205 E
Input user handle in parameter &1 not valid.

Chapter 3. Using Net.Data to Write CGI Programs for You

This chapter discusses Net.Data for AS/400.

Net.Data is an application that runs on a server and allows you to easily create dynamic web documents that are called web macros. Web macros that are created for Net.Data have the simplicity of HTML with the functionality of CGI-BIN applications. Net.Data makes it easy to add live data to static web pages. Live data includes information that is stored in databases, files, applications, and system services.

Overview of Net.Data

Net.Data is a comprehensive web development environment for the creation of simple dynamic web pages or complex web-based applications. These applications enable browser clients to access data from a variety of sources, such as databases, applications, and system services.

Net.Data consists of a program, the ***web macro processor***, and one or more dynamic libraries, called ***language environments***. The executable input to Net.Data is the ***web macro***.

The web macro processor communicates with IBM HTTP Server through its CGI-BIN interface. The server uses TCP/IP to connect to the Internet. Like other CGI-BIN programs, Net.Data is typically stored in the server's CGI-BIN directory. Net.Data is accessed when a URL received by the server refers to the web macro processor operable, DB2WWW, in the CGI-BIN directory.

Language environments are the web macro processor's interface to your data and applications. Each language environment provides a specific interface to a particular resource. For example, Net.Data provides language environments to access DB2® databases, REXX, and other applications via the SYSTEM language environment.

A web macro is a file that contains a series of statements that are defined by the Net.Data web macro language. These statements can include standard HTML and language environment-specific statements (for example, SQL statements) as well as macro directives. These statements act as instructions to the web macro processor, telling it how to construct dynamic web pages.

When a URL is received by the server that refers to the web macro processor program, the server starts an instance of the web macro processor. It then passes essential information, including the name of the requested web macro and the section of the macro to use. The web macro processor then:

1. Reads and parses through the web macro
2. Interprets all the macro statements, and
3. Dynamically builds the HTML page

When a web macro language %FUNCTION statement is encountered, the web macro processor loads the requested language environment-dynamic library (service program). It then passes language-specific information to the language

environment to be processed. The language environment processes the information and returns the results to the web macro processor.

After all parsing is done and language environment processing is completed, all that remains is pure HTML text. This text can then be interpreted by any browser. The web macro writer has complete control over the level of HTML it uses and what HTML tags are applied. The web macro processor imposes no restrictions. The pure HTML text is passed back to the server, and the web macro processor ends. The resulting HTML text is passed to the browser where the user interacts with it. Further requests from this user or any other user will result in the whole process just described taking place again.

For more detailed information about Net.Data, including how to configure Net.Data for the AS/400 and how to write Net.Data macros and language environments, see this URL:

http://www.as400.ibm.com/netdata

Chapter 4. Using Persistent CGI Programs

Overview of Persistent CGI

Persistent CGI is an extension to the CGI interface that allows a CGI program to remain active across multiple browser requests and maintain a session with that browser client. This allows files to be left open, the state to be maintained, and long running database transactions to be committed or rolled-back based on end-user input. The AS/400 CGI program must be written using named activation groups which allows the program to remain active after returning to the server. The CGI program notifies the server it wants to remain persistent using the ″Accept-HTSession″ CGI header as the first header it returns. This header defines the session ID associated with this instance of the CGI program and is not returned to the browser. Subsequent URL requests to this program must contain the session ID as the first parameter after the program name. The server uses this ID to route the request to that specific instance of the CGI program. The CGI program should regenerate this session ID for each request. It is strongly recommended that you use Secure Sockets Layer (SSL) for persistent and secure business transaction processing.

Named Activation Groups

CGI programs can be built using named activation groups by specifying a name on the ACTGRP parameter of the CRTPGM or CRTSRVPGM commands. In doing this, the initial call to the program within the job will still have the startup cost of activating the program. However, an activation group is left active after the program has exited normally. All storage associated with that program is still allocated and in ″last-used″ state. The program is not initializated when it is called again. In addition, for the ILE C runtime, all settings are in ″last-used″ state, such as signal(), strtok(). The RCLACTGRP command is used to end a named activation group. Use the DSPJOB OPTION(*ACTGRP) command to display all the activation groups for the job. All ILE languages running on AS/400 can use this mechanism to enable persistence for their CGI programs.

For additional information about activation groups see, *ILE Concepts*, SC41-5606 book.

Accept-HTSession CGI Header

This header specifies the session handle associated with this instance of the Persistent CGI program. This session handle is used to route back subsequent requests to that program and must be unique, or the server will not honor the persistence request. A message is logged in the error log of the server.

```
Accept-HTSession = "Accept-HTSession" ":" handle
```

When the server receives this header, the CGI job servicing the request will be reserved in a persistent state. Only requests coming in with that session handle in the URL are routed back to that instance of the CGI program. The URL must be in the following format:

```
/path/cgi-name/handle/rest/of/path
```

Where handle is an exact match of the handle provided in the ″Accept-HTSession″ CGI header for the program cgi-name.

Note: The cgi-name that is being resolved is the name as it appears in the URL. It is not necessarily the actual name of the program being started on the system. This is to remain consistent with the name resolution performed by the server.

HTTimeout CGI Header

The HTTimeout header is for the CGI program to define the amount of time, in minutes, that this CGI program wants to wait for a subsequent request. If not specified, the value specified on the PersistentCGITimeout directive is used. If specified, it takes precedence over the PersistentCGITimeout directive, but the server will not wait longer than the time specified on the MaxPersistentCGITimeout directive. This allows individual CGI programs to give users more time to respond to lengthy forms or explanations. However, it still gives the server ultimate control over the maximum time to wait.

```
HTTimeout = "HTTimeout" ":" minutes
```

The time-out value is a non-negative decimal integer, representing the time in minutes. This header must be preceded by an ″Accept-HTSession″ header, if not, it is ignored. If you omit the header, the default time-out value for the server is used. When a CGI program is ended because of a time-out, a message is logged in the error log of the server.

Considerations for using Persistent CGI Programs

You should be aware of the following considerations when using persistent CGI programs:

- The web administrator can limit the number of persistent CGI programs that the server supports by using the MaxPersistentCGI configuration directive.
- There are some job or thread-level resources that the server code running in the CGI job usually manipulates (directly or indirectly) on behalf of CGI programs. The following attributes will (potentially) change across calls:
  - Environment variables the server sets
  - Stdin/Stdout/Stderr file descriptors
  - User profile
  - Library list
- The server will not set the rest of the job attributes set by the server, and therefore, will maintain state across calls if changed by the CGI program. Note, however, that the CGI program must restore the initial state of these values before ending its persistence in order to guarantee compatibility across subsequent server requests:
  - Job Language, Country, CCSID
  - Job Priority
  - Printer/Output Queue
  - Message Logging
  - Environment variables set by the CGI program
- For added security, web server administrators can protect their persistent CGI programs using registered Internet users, thereby forcing authentication by the user before processing each request.

Persistent CGI Program Example

The following example shows a counter that is increased each time the Persistent CGI program is called.

```
/*********************************************************************/
/   This is a sample Persistent CGI program                           */
/   This program is invoked by a URL                                  */
/    http://hostname/cgi-bin/samplePersistent.pgm?bin=1)              */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFSIZE 1024
unsigned int MAXINPUT = BUFSIZE; /* Maximum input data.*/

int count=1;

int main()
{

char *pt;
char carac[2]="  ";
int bin=1;

freopen("", "r", stdin); /* You need to re-open the stdin for the Persistent CGI Program

pt=getenv("QUERY_STRING");
carac[0]=pt[4];
bin=atoi(carac);

if(bin == 1)
{
printf("Accept-HTSession: webpg101101 \n");
printf("Content-type: text/html \n\n");
printf("<html><title>Test persistent CGI</title><body>");
printf("<h2> The first form</h2>");
printf("<form action=\"/cgi-bin/webpg101.pgm/webpg101101\" ");
printf("method=\"GET\"> ");
printf("<input type=HIDDEN NAME=BIN VALUE=2>");
printf("<input type=reset value=Reset>");
printf("</form></body></html>");
count++;
}
if(bin == 2)
{
pt=getenv("QUERY_STRING");
printf("Accept-HTSession: webpg101101 \n");
printf("Content-type: text/html \n\n");
printf("<html><title> Test persistent CGI</title><body>");
printf("<h2> The second form</h2>");
printf(" Valor count: %i",count);
printf("Query string: %s",pt);
printf("<form action=\"/cgi-bin/webpg101.pgm/webpg101101\" ");
printf("method=\"GET\"> ");
printf("<input type=HIDDEN NAME=BIN VALUE=3>");
printf("<h2>Persisten CGI si funcionan</h2>");
printf("<input type=submit value=Execute>");
printf("</form></body></html>");
count++;
}
if(bin == 3)
{
printf("Accept-HTSession: webpg101101 \n");
printf("Content-type: text/html \n\n");
printf("<html><title> Test persistent CGI</title><body>");
printf("<h2> The third form</h2>");
```

```
printf(" Valor count: %i",count);
printf("<form action=\"/cgi-bin/webpg101.pgm?bin=4/webpg101101\" ");
printf("method=\"GET\"> ");
printf("<h2>Persisten CGI si funcionan</h2>");
printf("<input type=HIDDEN NAME=BIN VALUE=4>");
printf("<input type=submit value=Execute>");
printf("</form></body></html>");
count++;
}
if(bin == 4)
{
printf("Accept-HTSession: webpg101101 \n");
printf("Content-type: text/html \n\n");
printf("<html><title> Test persistent CGI</title><body>");
printf("<form action=\"/cgi-bin/webpg101.pgm?bin=5/webpg101101\" ");
printf("method=\"GET\"> ");
printf("<h2> The fourth form</h2>");
printf(" Valor count: %i",count);
printf("<h2>Persisten CGI si funcionan</h2>");
printf("<input type=HIDDEN NAME=BIN VALUE=5>");
printf("<input type=submit value=Execute>");
printf("</form></body></html>");
count++;
}
if(bin == 5)
{
printf("Accept-HTSession: webpg101101 \n");
printf("Content-type: text/html \n\n");
printf("<html><title> Test persistent CGI</title><body>");
printf("<form action=\"/cgi-bin/webpg101.pgm?bin=6/webpg101101\" ");
printf("method=\"GET\"> ");
printf("<h2> The fifth form</h2>");
printf(" Valor count: %i",count);
printf("<h2>Persisten CGI si funcionan</h2>");
printf("<input type=HIDDEN NAME=BIN VALUE=6>");
printf("<input type=submit value=Execute>");
printf("</form></body></html>");
count++;
}
(bin == 6)
{
printf("Accept-HTSession: webpg101101 \n");
printf("Content-type: text/html \n\n");
printf("<html><title> Test persistent CGI</title><body>");
printf("<h2> The sixth form</h2>");
printf(" Valor count: %i",count);
printf("<h2>Persisten CGI si funcionan</h2>");
printf("</body></html>");
}
fflush(stdout);
return 0;
}
```

Chapter 5. Enabling your AS/400 to run CGI programs

This chapter discusses the specific steps you need to take to enable your AS/400 for Common Gateway Interface (CGI) programs.

How to enable the server to run CGI programs

AS/400 stores some CGI programs in QSYS.LIB. You can write the programs in C++, Rexx, Java, ILE-C, RPG, or COBOL. If the UserID directive is *not* active, the server profile QTMHHTP1 needs access to the *PGM object and all objects the program accesses. If the UserID directive *is* active, the UserID profile needs access to the *PGM object and all objects the program accesses. The Exec directive is required in the HTTP configuration to run a CGI program on the server.

Here is a summary of the steps you need to take to enable your AS/400 system to run CGI programs:

1. Decide for which CGI mode you will write your program.
2. Write the C++, Rexx, Java, ILE-C, RPG, or COBOL program.
3. Compile your program.
4. Create the program object using CRTPGM. Add the Bind Service program, QHTTPSVR/QZHBCGI when the program uses the server APIs (QtmhWrStOut, QtmhRdStdIn, QtmhCvtDB, QtmhGetEnv, QtmhPutEnv QzhbCgiParse, or QzhbCgiUtils).
5. Using the WRKHTTPCFG command, add an Exec directive that either specifies the actual library where the program is stored or maps to the library where the program is stored. Specify the CGI mode for your program. The following directive is the library where the program is stored and also indicates to the server to use EBCDIC mode:

   ```
   Exec /QSYS.LIB/nnnnnnnn.LIB/* %%EBCDIC%%
   ```

   Where *nnnnnnnn* is the library where the CGI program is stored.

   The following directive maps to the library where the program is stored.

   ```
   Exec /CGI-BIN/* /QSYS.LIB/nnnnnnnn.LIB/* %%EBCDIC%%
   ```

   The advantage of using the mapping directives is that the actual location of documents and programs is masked. Also, by setting the `/cgi-bin` values correctly for Pass, Exec and Redirect, there is less chance of finding the wrong directive.

   Access to program object is *USE for QTMHHTP1 or *PUBLIC. You must set *USE for QTMHHTP1 for the access to the program object, or you must specify a user ID on a Userid directive in the server configuration. Setting the access to *PUBLIC *USE would enable the server to run the CGI program, regardless of whether you specify a user ID in the server configuration.
6. Store the HTML file on the AS/400 system by doing one of the following, depending on the file system in which you wish to store the document:
   - To store it in the AS/400 Source physical file:

a. Add a Pass directive using the WRKHTTPCFG command.
```
Pass /sample /qsys.lib/samplel.lib/samplef.file/samplem.mbr
```
Where *samplel* is the library, *samplef* is the file, and *samplem* is member name in which the AS/400 stores this document.
b. Set the source type of *samplem* member to HTML(CHGPFM).
Access to file is *USE for QTMHHTTP or *PUBLIC(GRTOBJAUT).

- To store it in the Integrated File System webtest directory:
  a. Add a Pass directive using the WRKHTTPCFG command.
  ```
  Pass /sample /webtest/sample.html
  ```
  where *webtest* is an integrated file system directory, and *sample.html* is the document.
  b. Access to file is *R for QTMHHTTP or *PUBLIC(CHGAUT).
- To store it in IFS QOpenSys/webtest directory:
  a. Add a Pass directive using the WRKHTTPCFG command.
  ```
  Pass /sample /qopensys/webtest/sample.html
  ```
  where *webtest* is an integrated file system directory in QOpenSys file system.
  b. Access to file is *R for QTMHHTTP or *PUBLIC(CHGAUT).
- To store it in the QDLS folder:
  a. Add a Pass directive using the WRKHTTPCFG command.
  ```
  Pass /sample /qdls/webtest/sample.htm
  ```
  Where *webtest* is a QDLS folder, and *sample.htm* is the document.
  b. Access to file is *R for QTMHHTTP or *PUBLIC(CHGAUT).

7. Add QTMHHTTP to AS/400 directory entry(WRKDIRE).
8. Finally, still using WRKHTTPCFG, enable POST in your HTTP configuration file. POST must be enabled in order for the server to serve CGI programs that read standard input.
9. Start or restart the server.
10. Point your web browser to the URL for the HTML document on the server where *hostname* is the fully qualified host domain name of your AS/400 system.
```
http://hostname/sample
```

Note: For REXX programs, you only need to indicate in the EXEC directive the path and the file name. REXX CGI execs must reside in database files named REXX or QREXSRC. For example:
```
EXEC /rexx/* /QSYS.LIB/AS400CGI.LIB/QREXSRC.FILE/*
```
The URL is :
```
http://hostname/rexx/samplecgi.rexx
```

Using directives for security and access control

The server administrator controls the behavior of the server. The server will not do anything that the server administrator has not explicitly configured it to do.

Several features of the server ensure that the administrator maintains this control:

- The default fail rule means that only requests that are authorized by the web administrator are honored; other requests will fail.
- Explicit CGI enablement means that no CGI programs will run unless specifically authorized
- Only CGI programs are run
- Only the read HTTP methods GET, POST, and HEAD are supported

The default fail rule

The server rejects, by default, all incoming requests unless the URL, as translated by any preceding Map directives, matches a Pass, Redirect, or Exec directive that has been explicitly coded by the server administrator:

- A match with a Pass directive enables the server to serve a document.
- A match with a Redirect directive causes the server to return a 302 response, found in the HTTP response to the client application. This HTTP response header field contains a location with the redirect request. The HTTP request that matches a Redirect directive causes no data to be accessed. A subsequent request generated by a client could cause data to be accessed.
- A match with an Exec directive enables the server to run a CGI program on behalf of the client.
- A match with a Service directive enables the server to run a server API program on behalf of the client.

Explicit CGI enablement

The server will not run a user-defined CGI program unless the server administrator has explicitly enabled it by coding an Exec directive. The server administrator can, for example, limit CGI requests to a specific library in QSYS.LIB.

Important!

It is the server administrator's responsibility to verify that any CGI program that is enabled does not violate the customer's security policies for the AS/400 system on which the server is running.

IBM recommends that the HTTP administrator move the DB2WWW *PGM (the Net.Data CGI program) from the QHTTPSVR library to its own CGI library. This allows users to run the CGI program while limiting access to the QHTTPSVR library. **Do not move any Include files from the QHTTPSVR library.**

Server runs only CGI programs

To run properly, programs that are called by the server must conform to the server CGI interface. When the server is enabled to call a particular program on behalf of a remote HTTP client application, the program is called and the output is returned through the server CGI interface.

CGI program considerations

You need to understand that the security environment defined by the server configuration directives that apply to your CGI programs.

If the CGI program is covered by a protection directive that calls for basic authentication, the user must supply a user ID and password before the CGI program is allowed to run. The other protection subdirectives determine the following:

- How the server validates the user ID and password
- What security environment the CGI program runs in

The subdirectives might tell the browser to treat the user ID as an AS/400 user profile and to validate the password against it. In addition, the Userid subdirective might be used to cause the server job to run under a specified AS/400 user profile or the one the user entered. The following example protection setup would cause the user ID to be treated as an AS/400 user profile, and to switch to that profile when starting the CGI program:

```
Protection    example1 {
  AuthType       Basic
  Userid         %%CLIENT%%
  PasswdFile     %%SYSTEM%%
}
```

If Userid %%SERVER%% had been specified, the CGI program will run under the QTMHHTP1 user profile. If Userid FRED had been specified, the CGI program would run under the FRED user profile.

Alternatively, the PasswdFile subdirective can identify a validation list. For example:

```
PasswdFile     qgpl/valist1
```

Validation lists contain a set of user IDs, their associated password, and optionally other application-specific information. In this example, the server would authenticate the user by comparing the specified user ID and password against the specified validation list. If the user ID exists in the validation list and the password matches, the CGI program would run under the QTMHHTP1 user profile.

Validation lists can be created through the CRTVLDL command. CGI or other programs can add, remove, find, or change entries through a set of APIs documented in the programming topic in the AS/400 Information Center. By using validation lists, the CGI program can “register” users and associate other information with each entry while at the same time using the basic authentication functions of the HTTP server to authenticate requests.

Chapter 6. Sample programs (in Java, C, and RPG)

This chapter contains samples of coding in Java, C, and RPG languages.

You can locate other programming samples through the following uniform resource locator (URL):

http://www.as400.ibm.com/tstudio/index.htm

Example of Java language CGI program

The samplejava program takes environmental and form variables and displays them back to the browser.

```
import java.io.DataInputStream;
import java.util.Hashtable;
import java.util.StringTokenizer;

class samplejava
{

         int x;
         int index;
         Hashtable cgi_vars = null;

         // String table with all the Environment Variables

         String[] EnvVar = { "GATEWAY_INTERFACE",
                         "SERVER_NAME",
                         "SERVER_SOFTWARE",
                         "SERVER_PROTOCOL",
                         "SERVER_PORT",
                         "PATH_INFO",
                         "PATH_TRANSLATED",
                         "SCRIPT_NAME",
                         "DOCUMENT_ROOT",
                         "REMOTE_HOST",
                         "REMOTE_ADDR",
"AUTH_TYPE",
                         "REMOTE_USER",
                         "REMOTE_IDENT",
                         "HTTP_FROM",
                         "HTTP_ACCEPT",
                         "HTTP_USER_AGENT",
                         "HTTP_REFERER",
                         "REQUEST_METHOD",
                         "CONTENT_TYPE",
                         "CONTENT_LENGTH",
                         "QUERY_STRING"};
```

```
samplejava()
{

        String userMethod;
        String cl;
        cl = new String();
        // Get the REQUEST_METHOD variable  (POST or GET)
        userMethod = System.getProperty("REQUEST_METHOD");

        if (userMethod != null)
        {
           if (userMethod.equalsIgnoreCase("POST"))
           {
             System.out.println("Server method not supporting");
             System.exit(0);
            }
           else
           {
               // if the method is GET
               if (userMethod.equalsIgnoreCase("GET"))
               {
                   // Get the Value of Query String
                   cl = System.getProperty("QUERY_STRING");
               }
               else
               {
                  errMsg("Invalid REQUEST_METHOD specified");
                  System.exit(0);
                }
            }
         }
         else
         {
           // Print No method
           errMsg ("No REQUEST_METHOD specified");
           System.exit(0);
          }

          if (cl == null )
          {
                 errMsg ("No user data");
                 System.exit(0);
           }
           else
           {
                  // fill the Hash table with the user values
                  cgi_vars = parseArguments(cl);
           }
      }
```

```
  private Hashtable parseArguments (String query_string)
  {
     Hashtable cgi_vars = new Hashtable();

     // get the first token scan for the '&' char
     StringTokenizer stringToken = new StringTokenizer(query_string,"&");

     while (stringToken.hasMoreTokens())
     {
      index++;
      // Split the first token into Variable and Value
      StringTokenizer subToken = new StringTokenizer(stringToken.nextToken(),"=");

      // Remove the '+' char from the Variable
      String variable = plussesToSpaces(subToken.nextToken());

      // Remove the '+' char from the Value
      String value = plussesToSpaces(subToken.nextToken());

      // Create the Keys to store the Variables and the Values in the Hash Table
      // the keys will be variable1, value1, variable2, value2, and so forth
      String temp1= new String("variable"+index);
      String temp2= new String("value"+index);

      // Store the variables and the values in the Hash table
      cgi_vars.put(temp1,translateEscapes(variable));
      cgi_vars.put(temp2,translateEscapes(value));

     }
     return cgi_vars;
  }
private String plussesToSpaces(String query_string)
     {

       // Substitute the '+' char to a blank char
       return query_string.replace('+', ' ');

     }

  private String translateEscapes(String query_string)
  {
    int percent_sign = query_string.indexOf('%');
    int ascii_val;
    String next_escape=null;
    String first_part=null;
    String second_part=null;

    while (percent_sign != -1)
    {
     next_escape = query_string.substring(percent_sign + 1, percent_sign + 3);
     ascii_val = (16 * hexValue(next_escape.charAt(0)) + hexValue(next_escape.charAt(1)));
     first_part = query_string.substring(0, percent_sign);
     second_part = query_string.substring(percent_sign + 3, query_string.length());
     query_string = first_part + (char)ascii_val + second_part;
     percent_sign = query_string.indexOf('%', percent_sign + 1);
    }
    return query_string;
```

```
 private int hexValue(char c)
    {
    int rc;

    switch(c)
         {
          case '1':
               rc = 1;
               break;
          case '2':
               rc = 2;
               break;
          case '3':
               rc = 3;
               break;
          case '4':
               rc = 4;
               break;
          case '5':
               rc = 5;
               break;
          case '6':
               rc = 6;
               break;
          case '7':
               rc = 7;
               break;
          case '8':
               rc = 8;
               break;
          case '9':
               rc = 9;
               break;
          case 'a':
          case 'A':
               rc = 10;
               break;
          case 'b':
          case 'B':
               rc = 11;
               break;
          case 'c':
          case 'C':
               rc = 12;
               break;
          case 'd':
          case 'D':
               rc = 13;
               break;
          case 'e':
          case 'E':
               rc = 14;
               break;
          case 'f':
          case 'F':
               rc = 15;
               break;
          default:
               rc = 0;
               break;
          }
return rc;
}
```

```
 private void errMsg(String message)
    {

    System.out.println("Content-type: text/html\n");
    System.out.println("<html>");
    System.out.println("<head>");
    System.out.println("<title>Error</title>");
    System.out.println("</head>");
    System.out.println("<body>");
    System.out.println("<h1>Error</h1>");
    System.out.println("<hr>");
    System.out.println("<p>");
    System.out.println("An internal error occurred.");
    System.out.println("The specific error message is shown below:");
    System.out.println("<p><pre>" + message + "</pre><p>");
    System.out.println("<p>");
    System.out.println("<hr>");
    System.out.println("</body>");
    System.out.println("</html>");
     }

private void display()
{
       System.out.println("Content-type: text/html\n");
       System.out.println("<html>");
       System.out.println("<head>");
       System.out.println("<title>Environment and User Variables</title>");
       System.out.println("</head>");
       System.out.println("<body>");
       System.out.println("<h1>Environment and User variables</h1>");
       System.out.println("<h2>Environment Variables</h2>");
       System.out.println("<table>");

       for (int i=0; i<22; i++)
           {
                if (System.getProperty(EnvVar[i]) != null)
                {
                  System.out.println("<tr>");
                {
                  System.out.println("<tr>");
                  System.out.println("<td align=right>" + EnvVar[i] + " : ");
                  System.out.println("<td>" + System.getProperty(EnvVar[i]));
                }
                else
                {
                    System.out.println("<tr>");
                    System.out.println("<td align=right>" + EnvVar[i] + " : ");
                    System.out.println("<td>NONE");
                }
           }
       System.out.println("</table>");
       System.out.println("</body>");
       System.out.println("</html>");
       System.out.flush();
       }

       public static void main (String args[])
       {
           samplejava cgi = new samplejava();
           cgi.display();
       }
}
```

Example of C language CGI program

To call the SAMPLEC C program, add the following lines to an HTML form:

```
<form method="POST" action="/CGI-BIN/SAMPLEC.PGM">
<input name="YourInput" size=42,2>
<br>
Enter input for the C sample and click <input type="SUBMIT" value="ENTER">
<p>The output will be a screen with the text,
"YourInput=" followed by the text you typed above.
The contents of environment variable SERVER_SOFTWARE is also displayed.
</form>
```

The SAMPLEC program shows how to write a CGI program in C language.

```
/*********************************************************************/
/*                                                                   */
/* Source File Name: QCSRC.SAMPLEC                                   */
/*                                                                   */
/* Module Name: SAMPLEC                                              */
/*                                                                   */
/* This sample code is provided by IBM for illustrative purposes only.*/
/* It has not been fully tested. It is provided as-is without any    */
/* warranties of any kind, including but not limited to the implied  */
/* warranties of merchantability and fitness for a particular purpose.*/
/*                                                                   */
/* Source File Description: A sample of a C program executed as a    */
/* CGI program on the AS/400 HTTP server.  The program demonstrates  */
/* reading standard input, reading environment variables and         */
/* writing standard output.  The input data comes from information   */
/* typed in HTML  fields.  This program will read this input    */
/* information and write exactly what is read to standard output.    */
/*                                                                   */
/* This program is a simple AS/400 ILE/C program that demonstrates   */
/* the ILE/C function calls for reading standard input, reading      */
/* environment variables and writing standard output.  The fread()   */
/* and printf() are used to read standard input and write standard   */
/* output.  getenv() is used to read HTTP server environment         */
/* variables. The fread() and printf() are found in stdio.h header   */
/* file and getenv() is found in stdlib.h.  These includes are needed */
/* in any AS/400 ILE/C program reading standard input, writing       */
/* standard output and reading environment variables.                */
/*                                                                   */
/* This program will write HTML type data to standard output.  The   */
/* first line of data written to standard output must be included and */
/* consists of the Content Type of the data that follows.            */
/*                                                                   */
/* Here are the steps to setting up and running this CGI program on  */
/* the AS/400 HTTP server:                                           */
/*                                                                   */
/* 1-> Create HTML document as a member in a source physical file.   */
/* 2-> Set Source type of HTML document to HTML.                     */
/* 3-> Create the *PGM object called SAMPLEC(CRTCMOD and CRTPGM).    */
/* 4-> Check QTMHHTTP or *PUBLIC has access to document and QTMHHTP1  */
/*       or *PUBLIC has access to program(DSPOBJAUT and GRTOBJAUT).  */
/*     (If configuration runs CGI request using Userid, The User     */
/*     profile specified on Userid directive must have access to     */
/*     program.)                                                     */
/* 5-> Set up HTTP server configuration directives(WRKHTTPCFG)       */
/* 6-> Start the HTTP server instance(STRTCPSVR).                    */
/* 7-> Request the document from the browser.                        */
/*                                                                   */
/* Sample the html form segment to run this sample CGI script:       */
/*                                                                   */
/*           .                                                       */
/*           .                                                       */
/*           .                                                       */
```

```
/* <form action="/cgi-bin/samplec" method="POST">                     */
/* <input type="Text" maxsize=80 size=20 name="Sample">               */
/* <input type="Submit" name="SampleData" value="Submit CGI script">  */
/* <br>Sample CGI program written in AS/400 ILE/C.<br>                */
/* Type data and Click Submit to run a Sample program using ILE/C.    */
/* </form>                                                            */
/*           .                                                        */
/*           .                                                        */
/*           .                                                        */
/* Change the form method to "GET" when input data for sample CGI     */
/* program comes from the QUERY_STRING environment variable.          */
/*                                                                    */
/* HTTP Server configuration:                                         */
/* # Pass the html document in a library                              */
/* Pass /sampledoc /qsys.lib/websamp.lib/htmlfile.file/samplec.mbr    */
/* # Allow CGI program to run.                                       */
/* Map  /cgi-bin/* /cgi-bin/*.pgm                                    */
/* Exec /cgi-bin/* /qsys.lib/websamp.lib/*                            */
/*                                                                    */
/* This program is invoked by a URL from a browser in the form:       */
/* http://hostname/sampledoc                                          */
/*                                                                    */
/* Functions tested: Calls to Standard input and output using         */
/*                   standard i/o c calls.  This program uses         */
/*                   fread(), putchar(), and printf(). It is          */
/*                   designed to run on the AS/400 HTTP Server.       */
/*                                                                    */
/**********************************************************************/


#include <stdio.h>                    /* C-stdio library.             */
#include <string.h>                   /* string functions.            */
#include <stdlib.h>                   /* stdlib functions.            */
#include <errno.h>                    /* errno values.                */


#define LINELEN  80                   /* Max length of line.          */



/**********************************************************************/
/*                                                                    */
/* Function Name: writeData()                                         */
/*                                                                    */
/* Descriptive Name: Function is used to print the data to the        */
/* browser.  The data is printed 80 characters/line to provide        */
/* a neat and readable output.                                        */
/*                                                                    */
/* HTTP Server Environment variables:                                 */
/* ----------------------------------                                 */
/*                                                                    */
/* Standard Input:                                                    */
/* ---------------                                                    */
/*                                                                    */
/* Standard Output:                                                   */
/* ----------------                                                   */
/* All data directed to Standard output is sent using printf() or     */
/* putchar().  Standard output is written with html text.             */
/*                                                                    */
/*                                                                    */
/* Input: ptrToData : A pointer to the data to write to stdout.       */
/*        dataLen   : Length of data buffer.                          */
/*                                                                    */
/* Output: Data buffer written to stdout.                             */
/*                                                                    */
/* Exit Normal:                                                       */
/*                                                                    */
```

```
/* Exit Error: None                                                   */
/*                                                                    */
/**********************************************************************/
void writeData(char* ptrToData, int dataLen)
{

  div_t  insertBreak;
  int    i;

  /*-------------------------------------------------------------------*/
  /* Write dataLen bytes of data from ptrToData.                       */
  /*-------------------------------------------------------------------*/
  for (i=1; i<= dataLen; i++) {

      putchar(*ptrToData);
      ptrToData++;

      /*---------------------------------------------------------------*/
      /* Print a break after every 80 characters.                      */
      /*---------------------------------------------------------------*/
      insertBreak = div(i, LINELEN);
      if ( insertBreak.rem == 0 )
          printf("<br>");

  }

  return;

}
```

```
/**********************************************************************/
/*                                                                    */
/* Function Name: main()                                              */
/*                                                                    */
/* Descriptive Name: A sample of the method used for AS/400 ILE/C to  */
/* read standard input, write standard output and check environment   */
/* variables; SERVER_SOFTWARE, REQUEST_METHOD, CONTENT_LENGTH, etc.   */
/*                                                                    */
/* HTTP Server Environment variables:                                 */
/* ----------------------------------                                 */
/* The C function call, getenv, is used to read AS/400 server         */
/* environment variables.  The value of the argument is a (char *)    */
/* pointer with the name of the environment variable. The value of    */
/* the environment variable is always returned as a string pointer.   */
/* The value may need to be converted to be used; that is             */
/* CONTENT_LENGTH needs to be converted to int using atoi().          */
/*                                                                    */
/* Standard Input:                                                    */
/* ---------------                                                    */
/* CONTENT_LENGTH is used to determine the amount of data to be       */
/* read from standard input with fread().  The standard input is      */
/* considered to be a stream of bytes up to CONTENT_LENGTH bytes. The */
/* standard input can be read with any file input stream function up  */
/* to and including CONTENT_LENGTH bytes.  Reading more than          */
/* CONTENT_LENGTH bytes is not defined.                               */
/*                                                                    */
/* Standard Output:                                                   */
/* ----------------                                                   */
/* All data directed to Standard output is using writeData().         */
/*                                                                    */
/* Standard output is written with html text which includes HTTP      */
/* header lines identifying the content type of the data written and  */
/* HTTP response headers.  This MUST be followed by a blank line(\n\n)*/
```

```
/* before writing any html text.  This indicates the end of the       */
/* header and the start of text that is served from the server.       */
/* This text is usually html but can be plain/text.                   */
/*                                                                    */
/* Input: Data read from standard input or QUERY_STRING that is       */
/*        entered in an HTML form.                                    */
/*                                                                    */
/* Output: The data read from standard input is written as is to      */
/*         standard output.  This information would then be served by */
/*         the HTTP server.                                           */
/*                                                                    */
/* Exit Normal:                                                       */
/*                                                                    */
/* Exit Error: None                                                   */
/*                                                                    */
/********************************************************************/
void main()
{

  char   *stdInData;                  /* Input buffer.              */
  char   *queryString;                /* Query String env variable  */
  char   *requestMethod;              /* Request method env variable */
  char   *serverSoftware;             /* Server Software env variable*/
  char   *contentLenString;           /* Character content length.  */
  int    contentLength;               /* int content length         */
  int    bytesRead;                   /* number of bytes read.      */
  int    queryStringLen;              /* Length of QUERY_STRING     */

  /*------------------------------------------------------------------*/
  /* The "Content-type" is the minimum request header that must be    */
  /* written to standard output.  It describes the type of data that  */
  /* follows.                                                         */
  /*------------------------------------------------------------------*/
  printf("Content-type: text/html\n");

  /*------------------------------------------------------------------*/
  /* VERY IMPORTANT! An extra newline must be written                 */
  /* after the request header.  In this case the request header is    */
  /* only the Content-type.  This tells the HTTP server that the      */
  /* request header is ended and the data follows.                    */
  /*------------------------------------------------------------------*/
  printf("\n");

  /*------------------------------------------------------------------*/
  /* This html text consists of a head and body section.  The head    */
  /* section has a title for the document.  The body section will     */
  /* contain standard input, QUERY_STRING, CONTENT_LENGTH,            */
  /* SERVER_SOFTWARE and REQUEST_METHOD.                              */
  /*------------------------------------------------------------------*/
  printf("<html>\n");
  printf("<head>\n");
  printf("<title>\n");
  printf("Sample AS/400 HTTP Server CGI program\n");
  printf("</title>\n");
  printf("</head>\n");
  printf("<body>\n");
  printf("<h1>Sample AS/400 ILE/C program.</h1>\n");
  printf("<br>This is sample output writing in AS/400 ILE/C\n");
  printf("<br>as a sample of CGI programming.  This program reads\n");
  printf("<br>the input data from Query_String environment\n");
  printf("<br>variable when the Request_Method is GET and reads\n");
  printf("<br>standard input when the Request_Method is POST.\n");

  /*------------------------------------------------------------------*/
  /* Get and write the REQUEST_METHOD to stdout.                      */
  /*------------------------------------------------------------------*/
  requestMethod = getenv("REQUEST_METHOD");
```

```
if ( requestMethod )
    printf("<h4>REQUEST_METHOD:</h4>%s\n", requestMethod);
else
    printf("Error extracting environment variable REQUEST_METHOD.\n");

/*--------------------------------------------------------------------*/
/* html form data can be provided to the CGI program either on        */
/* stdin or in environment variable QUERY_STRING.  This can be        */
/* determined by examining REQUEST_METHOD.                            */
/*--------------------------------------------------------------------*/
if ( strcmp(requestMethod,"POST") == 0 ) {

    /*----------------------------------------------------------------*/
    /* The REQUEST_METHOD is "POST".  The environment variable        */
    /* CONTENT_LENGTH will tell us how many bytes of data to read     */
    /* from stdin.  Note: CONTENT_LENGTH must be convert to an int.   */
    /*----------------------------------------------------------------*/
    contentLenString = getenv("CONTENT_LENGTH");
    contentLength = atoi(contentLenString);

    /*----------------------------------------------------------------*/
    /* Write CONTENT_LENGTH to stdout.                                */
    /*----------------------------------------------------------------*/
    printf("<h4>CONTENT_LENGTH:</h4>%i<br><br>\n",contentLength);

    if ( contentLength ) {

        /*------------------------------------------------------------*/
        /* Allocate and set memory to read stdin data into.           */
        /*------------------------------------------------------------*/
        stdInData = malloc(contentLength);
        if ( stdInData )
            memset(stdInData, 0x00, contentLength);
        else
            printf("ERROR: Unable to allocate memory\n");
        /*------------------------------------------------------------*/
        /* A CGI program MUST read standard input as a stream         */
        /* file only up to and including CONTENT_LENGTH bytes.        */
        /* Never should a program read more than CONTENT_LENGTH       */
        /* bytes.  A CGI program that reads standard input must       */
        /* never depend on an end of file flag.  This will cause      */
        /* unpredictable results when the CGI program reads           */
        /* standard input.                                            */
        /*------------------------------------------------------------*/
        printf("<h4>Server standard input:</h4>\n");
        bytesRead = fread((char*)stdInData, 1, contentLength, stdin);

        /*------------------------------------------------------------*/
        /* If we successfully read all bytes from stdin, format and   */
        /* write the data to stdout using the writeData function.     */
        /*------------------------------------------------------------*/
        if ( bytesRead == contentLength )
            writeData(stdInData, bytesRead);
        else
            printf("<br>Error reading standard input\n");

        /*------------------------------------------------------------*/
        /* Free the storage allocated to hold the stdin data.         */
        /*------------------------------------------------------------*/
        free(stdInData);

    } else
        printf("<br><br><b>There is no standard input data.</b>");

} else if (strcmp(requestMethod, "GET") == 0 ) {
    /*----------------------------------------------------------------*/
    /* The REQUEST_METHOD is "GET".  The environment variable         */
```

```
      /* QUERY_STRING will contain the form data.                    */
      /*-------------------------------------------------------------*/
      queryString = getenv("QUERY_STRING");
      if ( queryString ) {

          /*---------------------------------------------------------*/
          /* Write the QUERY_STRING data to stdout.                  */
          /*---------------------------------------------------------*/
          printf("<h4>Server input read from QUERY_STRING:</h4>");
          queryStringLen = strlen(queryString);
          if ( queryStringLen )
              writeData(queryString, queryStringLen);
          else
              printf("<b>There is no data in QUERY_STRING.</b>");

      } else
          printf("<br>Error getting QUERY_STRING variable.");

  } else
      printf("<br><h2>ERROR: Invalid REQUEST_METHOD.</h2>");

  /*-----------------------------------------------------------------*/
  /* Write break and paragraph html tag to stdout.                   */
  /*-----------------------------------------------------------------*/
printf("<br><p>\n");

 /*-----------------------------------------------------------------*/
  /* Write the SERVER_SOFTWARE environment variable to stdout.       */
  /*-----------------------------------------------------------------*/
  serverSoftware = getenv("SERVER_SOFTWARE");
  if ( serverSoftware )
      printf("<h4>SERVER_SOFTWARE:</h4>%s\n", serverSoftware);
  else
      printf("<h4>Server Software is NULL</h4>");

  /*-----------------------------------------------------------------*/
  /* Write the closing tags on HTML document.                       */
  /*-----------------------------------------------------------------*/
  printf("</p>\n");
  printf("</body>\n");
  printf("</html>\n");

  return;
}
```

Example of RPG language CGI program

To call the SAMPLE RPG program, add the following lines to an HTML form:

```
<form method="POST" action="/CGI-BIN/SAMPLE.PGM">
<input name="YourInput" size=42,2>
<br>
Enter input for the RPG sample and click <input type="SUBMIT" value="ENTER">
<p>The output will be a screen with the text,
"YourInput=" followed by the text you typed above.
The contents of environment variable SERVER_SOFTWARE is also displayed.
</form>
```

The SAMPLE program shows how to write a CGI program in RPG language.

```
     ****************************************************************************
     ****                   Sample ILE RPG program.                          ***
     ****                                                                    ***
     ****This sample code is provided by IBM for illustrative purposes only.***
     ****It has not been fully tested. It is provided as-is without any     ***
     ****warranties of any kind, including but not limited to the implied   ***
     ****warranties of merchantability and fitness for a particular purpose.***
```

```
****                                                                    ***
**** This program is a simple RPG program that demonstrates the HTTP   ***
**** server APIs for reading standard input, reading an environment    ***
**** variable and writing standard output.  This is done using the     ***
**** IBM AS/400 HTTP Server APIs.                                       ***
****                                                                    ***
**** The HTML at the end of this listing in CTDATA HTML is the text    ***
**** that is modified by this program, written to standard output and  ***
**** served to a client.                                               ***
****                                                                    ***
**** 1-> Create HTML document as source physical file in library.      ***
**** 2-> Set Source type of HTML document to HTML.                     ***
**** 3-> Create the *PGM object called SAMPLE(CRTRPGMOD and CRTPGM).   ***
****      -Include the service program QTCP/QTMHCGI when doing the     ***
****        CRTPGM in the BNDSRVPGM or BNDDIR parameter.               ***
**** 4-> Check QTMHHTTP or *PUBLIC has access to document and QTMHHTP1 ***
****       or *PUBLIC has access to program(DSPOBJAUT and GRTOBJAUT).  ***
**** 5-> Set up HTTP server configuration directives(WRKHTTPCFG)       ***
**** 6-> Start the HTTP server(STRTCPSVR).                             ***
**** 7-> Run request from a browser.                                   ***
****                                                                    ***
****----------------------- IMPORTANT ---------------------------------***
**** The input for this program comes from CGI standard input or       ***
**** the environment variable, QUERY_STRING.  For an HTTP request      ***
**** method of POST, the input is read from standard input and         ***
**** for an HTTP request method of GET, the input is read from         ***
**** QUERY_STRING. For method POST, this program will only read 1024   ***
**** characters.  For method GET, the program will not read any        ***
**** input data when it exceeds 1024 characters.  The QtmhGetEnv       ***
**** API will set the length of the environment variable response      ***
**** to the actual length and no data is read into the receive         ***
**** buffer.                                                           ***
****----------------------- IMPORTANT ---------------------------------***
****                                                                    ***
**** Sample html form segment to run this sample CGI script:           ***
****                                                                    ***
****           .                                                        ***
****           .                                                        ***
****           .                                                        ***
**** <form action="/cgi-bin/sample" method="POST">                     ***
**** <input type="Text" maxsize=80 size=20 name="Sample">              ***
**** <input type="Submit name="SampleData" value="Submit CGI script">  ***
**** <br>Sample CGI program written in RPG.<br>                        ***
**** Type data and Click Submit to run a Sample program using RPG.     ***
**** </form>                                                           ***
****           .                                                        ***
****           .                                                        ***
****           .                                                        ***
****                                                                    ***
**** HTTP Server configuration:                                        ***
**** # Pass the html document in a library                            ***
**** Pass /sampledoc /qsys.lib/websamp.lib/htmlfile.file/sample.mbr    ***
**** # Allow CGI program to run.                                      ***
**** Map  /cgi-bin/* /cgi-bin/*.pgm                                   ***
**** Exec /cgi-bin/* /qsys.lib/websamp.lib/*                           ***
****                                                                    ***
**** This program is invoked by a URL from a browser in the form:      ***
**** http://hostname/sampledoc                                         ***
****                                                                    ***
************************************************************************
****                                                                    ***
**** Function of this SAMPLE program:                                  ***
**** Sample ILE RPG AS/400 program to demonstrate AS/400 HTTP server   ***
**** CGI program. It reads data from standard input based on the       ***
**** Content_Length environment variable.  The QtmhGetEnv System API   ***
**** is used to get the Content_Length and set the InDataLn variable   ***
**** used by the QtmhRdStdIn API. The data to be returned to the       ***
```

```
 **** client to written to standard output using the QtmhWrStOut API.   ***
 **** The data will be returned as text/html.                            ***
 ****                                                                   ***
 ************************************************************************
 * Variables for the CGI interface API for QtmhRdStIn.
DBufIn            S           1024a   INZ
DBufInLn          S              9b 0 INZ(1024)
DStdInLn          S              9b 0
 ************************************************************************
 * Variables for the CGI interface API for QtmhGetEnv.
DEnvRec           S           1024A   INZ
DEnvRecLen        S              9B 0 INZ(1024)
DEnvLen           S              9B 0 INZ
DEnvName          S             25A   INZ('CONTENT_LENGTH')
DEnvNameLen       S              9B 0 INZ(14)
 ************************************************************************
 *Variables for the CGI interface API for QtmhWrStout.
DBufOut           S           2048a   INZ
DBufOutln         S              9b 0
 ************************************************************************
 ***                  Data structure for error reporting.             ***
 *** Copied from QSYSINC/QRPGLESRC(QUSEC).                             ***
 *** The QUSBPRV must be initialized to 16.                            ***
 *** This is the common error structure that is passed to the CGI APIs;***
 *** QtmhWrStOut, QtmhRdStin, QtmhGetEnv and QtmhCvtDb.  The Error     ***
 *** structure is documented in the "AS/400 System API Reference".     ***
 ************************************************************************
DQUSEC            DS
D*                                          Qus EC
D QUSBPRV                 1      4B 0 INZ(16)
D*                                          Bytes Provided
D QUSBAVL                 5      8B 0
D*                                          Bytes Available
D QUSEI                   9     15
D*                                          Exception Id
D QUSERVED               16     16
 ************************************************************************
 *** Constants for names of CGI APIs.                                  ***
DAPIStdIn         C                   'QtmhRdStin'
DAPIStdOut        C                   'QtmhWrStout'
DAPIGetEnv        C                   'QtmhGetEnv'
 ************************************************************************
 * Prototype for c2n procedure that converts content length to numeric. ***
Dc2n              PR            30p 9
Dc                              32    options(*varsize)
 ************************************************************************
 * Compile-time array for HTML output.                                  ***
Darrsize          C                   23
Dhtml             S             80    DIM(arrsize) PERRCD(1) CTDATA
DContentLn        S              9B 0 INZ(0)
DEnvCL            S             20A   INZ('CONTENT_LENGTH')
DEnvSS            S             20A   INZ('SERVER_SOFTWARE')
DEnvMethod        S             20A   INZ('REQUEST_METHOD')
DEnvQS            S             20A   INZ('QUERY_STRING')
DEnvMDResp        S             30A   INZ
DEnvSSResp        S             50A   INZ
DEResp            S              4A   INZ
D************************************************************************
D* Define line feed that is required when writing data to std output.   ***
Dlinefeed         C                   x'15'
Dbreak            C                   '<br>'
Dmaxdataln        S              4B 0 INZ(1024)
D************************************************************************
D* Some local variables used for adding newline in std output buffer.   ***
Dcnt              S              4B 0 INZ(1)
DWORK2            S             80A   INZ
DResult           S              9B 0 INZ
```

```
 ****************************************************************************
 * Start of CGI Program execution section...
 ****************************************************************************
 * Initialize error code structure for error ids.
 * This allows for 7 bytes in QUSEI for error message id.
C                   Z-ADD     16            QUSBPRV
 ****************************************************************************
 **** Read the Environment variable, REQUEST_METHOD.
 ****************************************************************************
C                   MOVEL     EnvMethod     EnvName
C                   Z-ADD     14            EnvNameLen
C                   callb     APIGetEnv
C                   parm                    EnvRec
C                   parm                    EnvRecLen
C                   parm                    EnvLen
C                   parm                    EnvName
C                   parm                    EnvNameLen
C                   parm                    QUSEC
C                   MOVEL     EnvRec        EnvMDResp
 ****************************************************************************
 **** Is the REQUEST_METHOD, POST?
C     4             subst     EnvRec:1      EResp
C     EResp         ifeq      'POST'
 ****************************************************************************
 * Get Environment Variable 'Content_Length' using 'QtmhGetEnv' API
C                   MOVEL     EnvCL         EnvName
C                   Z-ADD     14            EnvNameLen
C                   CALLB     APIGetEnv
C                   parm                    EnvRec
C                   parm                    EnvRecLen
C                   parm                    EnvLen
C                   parm                    EnvName
C                   parm                    EnvNameLen
C                   parm                    QUSEC
 * Convert Content_Length to numeric.
C                   eval      ContentLn=c2n(EnvRec)
 * When the Content Length is greater than the buffer, Read maxdataln.
C     ContentLn     ifgt      maxdataln
C                   Z-ADD     maxdataln     ContentLn
C                   endif
 * Specify InDataLn to Content_Length value.  Never should a CGI program
 * ever attempt to read more than content length.  Specification of more
 * than content length in InDataLn is not defined.
C                   Z-ADD     ContentLn     BufInLn
 ****************************************************************************
 * Read standard input
C                   callb     APIStdIn
C                   parm                    BufIn
C                   parm                    BufInLn
C                   parm                    StdInLn
C                   parm                    QUSEC
C                   MOVEL     StdInLn       Result
C                   else
 ****************************************************************************
 **** Read the Environment variable, QUERY_STRING.
 ****************************************************************************
C                   MOVEL     EnvQS         EnvName
C                   Z-ADD     12            EnvNameLen
C                   callb     APIGetEnv
C                   parm                    EnvRec
C                   parm                    EnvRecLen
C                   parm                    EnvLen
C                   parm                    EnvName
C                   parm                    EnvNameLen
C                   parm                    QUSEC
 ****************************************************************************
 **** Check length of environment value is less than
```

```
 **** the receive buffer.  When this occurs, the
 **** QtmhGetEnv sets the EnvLen to the actual value
 **** length without changing the receive buffer.
C     EnvLen        ifgt      maxdataln
C                   eval      Bufin='Data buffer +
C                              not big enough for +
C                              available input data.'
C                   Z-ADD     80            Result
C                   else
C                   MOVEL     EnvRec        BufIn
C                   MOVEL     EnvLen        Result
C                   endif
C                   endif
 ***************************************************************************
 **** Read the Environment variable, SERVER_SOFTWARE.
 ***************************************************************************
C                   MOVEL     EnvSS         EnvName
C                   Z-ADD     15            EnvNameLen
C                   callb     APIGetEnv
C                   parm                    EnvRec
C                   parm                    EnvRecLen
C                   parm                    EnvLen
C                   parm                    EnvName
C                   parm                    EnvNameLen
C                   parm                    QUSEC
C                   MOVEL     EnvRec        EnvSSResp
 ***************************************************************************
 **** Put the data written to standard output in buffer; bufout.        ***
 ***************************************************************************
 * For each line of HTML, move it to BufOut and set the
 * output buffer's length(BufOutLn).
C                   do        arrsize       i                 5 0
 * Write out HTTP response and HTML lines.
C     i             iflt      17
C     BufOut        cat       html(i):0     BufOut
C     BufOut        cat       linefeed:0    BufOut
C                   endif
 * Add the data read from standard input or QUERY_STRING.
C     i             ifeq      17
D* Add html break to BufOut string written to standard output
D* when input is greater than 79.
C     Result        dowgt     79
C     80            SUBST     BufIn:cnt     WORK2
C                   cat       work2:0       BufOut
C                   cat       break:0       BufOut
 * For V4R2, the newline after 254 characters is not needed.
C*                  cat       linefeed:0    BufOut
C                   add       80            cnt
C                   sub       80            Result
C                   ENDDO
C                   IF        Result > 0
C                   clear                   WORK2
C     Result        SUBST     BufIn:cnt     WORK2
C                   cat       work2:0       BufOut
C                   cat       break:0       BufOut
 * For V4R2, the newline after 254 characters is not needed.
C*                  cat       linefeed:0    BufOut
C                   ENDIF
C                   endif
 * Add the Environment variable header line for REQUEST_METHOD.
C     i             ifeq      18
C     BufOut        cat       html(i):0     BufOut
C     BufOut        cat       break:0       BufOut
 * For V4R2, the newline after 254 characters is not needed.
C*    BufOut        cat       linefeed:0    BufOut
C                   endif
 * Display the Environment variable REQUEST_METHOD.
```

```
C     i             ifeq      19
C     BufOut        cat       EnvMDResp:0   BufOut
 * For V4R2, the newline after 254 characters is not needed.
C*    BufOut        cat       linefeed:0    BufOut
C                   endif
 * Add the Environment variable header line for SERVER_SOFTWARE.
C     i             ifeq      20
C     BufOut        cat       html(i):0     BufOut
C     BufOut        cat       break:0       BufOut
 * For V4R2, the newline after 254 characters is not needed.
C*    BufOut        cat       linefeed:0    BufOut
C                   endif
 * Display the Environment variable SERVER_SOFTWARE.
C     i             ifeq      21
C     BufOut        cat       EnvSSResp:0   BufOut
 * For V4R2, the newline after 254 characters is not needed.
C*    BufOut        cat       linefeed:0    BufOut
C                   endif
 * Write out closing HTML lines.
C     i             ifgt      21
C     BufOut        cat       html(i):0     BufOut
 * For V4R2, the newline after 254 characters is not needed.
C*    BufOut        cat       linefeed:0    BufOut
C                   endif
C                   enddo
 ****************************************************************************
 **** Get length of data to be sent to standard output.
 ****************************************************************************
C                   z-add     1             i
C     arrsize       mult      80            i
C     a             doune     ' '
C     1             subst     bufout:i      a                1
C                   sub       1             i
C                   enddo
C     i             add       1             BufOutLn
 ****************************************************************************
 **** Send BufOut to standard output.
 ****************************************************************************
C                   callb     APIStdOut
C                   parm                    BufOut
C                   parm                    BufOutLn
C                   parm                    QUSEC
 ****************************************************************************
 * Return to caller
 ****************************************************************************
C                   return
 ******************************************************
 * Function: Convert a character to numeric value.      *
 ******************************************************
 * nomain c2n subprocedure
Pc2n              B                   export
Dc2n              PI            30p 9
Dc                              32    options(*varsize)
 * variables
Dn                s             30p 9
Dwknum            s             30p 0
Dsign             s              1  0 inz(1)
Ddecpos           s              3  0 inz(0)
Dindecimal        s              1    inz('0')
Di                s              3  0
Dj                s              3  0
D                 ds
Dalpha1                          1
Dnumber1                         1  0 overlay(alpha1) inz(0)
C                   eval      c = %triml(c)
C     ' '           checkr    c             j
C     1             do        j             i
```

```
     C                   eval      alpha1=%subst(c:i:1)
     C                   select
     C                   when      alpha1='-'
     C                   eval      sign= -1
     C                   when      alpha1='.'
     C                   eval      indecimal='1'
     C                   when      alpha1 >='0' and alpha1 <= '9'
     C                   eval      wknum  = wknum  * 10 + number1
     C                   if        indecimal = '1'
     C                   eval      decpos = decpos + 1
     C                   endif
     C                   endsl
     C                   enddo
     C                   eval      n = wknum  * sign / 10 ** decpos
     C                   return    n
     Pc2n              E
      ***************************************************************************
      * Compile-time array follows:
      ***************************************************************************
      * A line MUST follow Content-type with only a single newline(x'15').  If
      * this newline does not exist, Then NO data will be served to the client.
      * This newline represents the end of the HTTP header and the data follows.
**CTDATA HTML
Content-type: text/html


<html>
<head>
<title>Sample AS/400 RPG program executed by HTTP Server as a CGI</title>
</head>
<body>
<h1>Sample AS/400 RPG program.</h1>
<br>
<br>
<p>This is sample output using AS/400 HTTP Server CGI APIs from an RPG
program.  This program reads the input data from Query_String
environment variable when the Request_Method is GET and reads
standard input when the Request_Method is POST.
<p>Server input:<br>

<p>Environment variable - REQUEST_METHOD:

<p>Environment variable - SERVER_SOFTWARE:

</body>
</html>
```

Example of a C language server configuration API program

```
/*

    This C source file is for compiling into the sample program
    APISAMPLE.PGM.  It invokes the new configuration file APIs
    contained in SRVPGM QHTTPSVR/QZHBCONF to read in
    a configuration file, and either replace an existing PORT
    directive or to add a new one.

    This code is written by IBM, and is intended only as a sample.
    There is no implied support for this code, and it is not
    a part of any IBM product.  It can be freely copied, modified
    and used in any way desired.

*/

#include <stdio.h>
```

```
#include <stdlib.h>

#include <qusec.h>      /* For errcode structure */
#include <qzhbconf.h>   /* For group file API's */

int main (int argc, char **argv)
{
    Qus_EC_t errcode;               /* Error code structure */
    unsigned char configname10[10]; /* Config name */
    unsigned int cfghdl;            /* Handle for config file */
    unsigned int getlock = 1;       /* Argument to request a write lock */
    unsigned char valstr[100];      /* Value string argument */
    unsigned int vallen;            /* Length argument */
    unsigned int numtofind = 0;     /* Will be searching for last directive */
    unsigned int casesense = 0;     /* Case insensitive search */
    unsigned int writecfg = 1;      /* Write config back out */
    unsigned int dirhdl;            /* Handle for a directive */

    if (argc <= 2 || strlen(argv[1]) > 10 || atoi(argv[2]) < 1) {
        printf("usage:  call lib/prog 'configname' 'portnumber'\n");
        return 1;
    }

    /* Get config name into a 10 character format */
    strncpy((char *) configname10, "          ", 10);
    strncpy((char *) configname10, argv[1], strlen(argv[1]));

    /* Set up error code structure */
    errcode.Bytes_Provided = sizeof(Qus_EC_t);

    /* Open the config - program will end if error occurs */
    QzhbOpenConfig(configname10, &getlock, &cfghdl, NULL);

    /* Search for the last PORT directive in the file */
    strcpy((char *) valstr, "port");
    vallen = strlen((char *) valstr);
    QzhbFindDirective(&cfghdl, valstr, &vallen, NULL,
                      &numtofind, &casesense, &dirhdl,
                     (unsigned char *) &errcode);

    /* Build string containing what we want the PORT directive to look like */
    sprintf((char *) valstr, "Port %s", argv[2]);
    vallen = strlen((char *) valstr);

    /* If found a PORT directive, replace it with our new value */
    if (errcode.Bytes_Available == 0) {
/* Replace existing directive, letting error end the program */
        QzhbReplaceDirective(&cfghdl, &dirhdl,
                             valstr, &vallen, NULL);
        printf("Replaced existing PORT directive in configuration %s with: %s\n",
               argv[1], valstr);
    }

    /* If did not find the PORT directive, we want to add a new one */
    else {
        unsigned int insertpos = 4;  /* Automatic positioning */
/* Add new directive, letting error end the program */
        QzhbAddDirective(&cfghdl, valstr, &vallen,
                         &insertpos, NULL, &dirhdl, NULL);
        printf("Added new PORT directive in configuration %s as: %s\n",
               argv[1], valstr);
```

```
    }

    /* Close config and write contents back out */
    QzhbCloseConfig(&cfghdl, &writecfg, NULL);

    return 0;
}
```

Chapter 7. Writing Server API programs

Overview of the Server API

The Server API allows you to extend the server's base functions. You can write extensions to do customized processing, such as:

- Enhance the basic authentication or replace it with a site-specific process.
- Add error handling routines to track problems or alert for serious conditions.
- Detect and track information that comes in from the requesting client, such as server referrals and user agent code.

General procedure for writing Server API programs

Before you start writing your Server API programs, you need to understand how the IBM HTTP Server works. The server performs a sequence of steps for each client request that it processes. Your program can run and make function calls at any of these steps. You need to decide where in the basic server request process you want to add customized functions. For example, do you want the server to do something after it reads a client request but before it performs any other processing? Or, maybe you want the server to perform special routines during authentication and then after it sends the requested file.

You can instruct the server to call the application functions in your program at the appropriate processing step. You can do this by using the API directives in your server configuration file.

Guidelines

Use the following guidelines when creating Server API programs:

- If compiling on AS/400 with Integrated Language Environment (ILE)/C, ensure that you include QHTTPSVR in the library list.
- Give each of your application functions a unique function name and call the server predefined functions as needed. Be sure to include HTAPI.h and to use the HTTPD_LINKAGE macro in your function definitions to avoid abending the server. This macro ensures that all functions use the same calling conventions.
- The server runs in a multithreaded environment; therefore, your application functions must be threadsafe. If your application is re-entrant, performance will not decrease.
- Keep the actions in your applications to a thread scope. Do not perform any actions as a process scope, such as exit or change user ID.
- Eliminate global variables or protect them with a mutual exclusion semaphore.
- Do not forget to set the Content-Type header if you are using HTTPD_write() to send data back to the client.
- You also must take into consideration the Content-Encoding header when you use HTTPD_write() to send data back to the client.
- Always check your return codes and provide conditional processing where necessary.
- Compile and then create your service program by using CRTSRVPGM. When using CRTSRVPGM, you must bind to QZHBIAPI *SRVPGM that is in the QHTTPSVR library.

- Add Server API directives to your configuration file so that you can associate your program's application functions with the appropriate step. There is a separate directive for each server request processing step. You must stop and restart the server for the new directives take effect.
- Test your program rigorously. Because IBM HTTP Server is a theaded server, you should apply more rigorous testing than you would for a forked server. Because the server calls your program directly and they both run in the same process space, errors in your program can stop the server.

Basic server request process

You can break down the basic server request process into a number of steps, based on the type of processing the server is performing during that phase. Each step includes a juncture at which a specified part of your program can run. The Server API directives in your configuration file, indicate which of your application functions you want called during a request process step.

Your compiled program exists as a service program. As the server proceeds through its request process steps, it calls the application functions associated with each step, until one of the functions indicates it has handled the request. If you have more than one of your application functions indicated for a particular step, your functions are called in the order in which they appear in the configuration file.

If the request is not handled by an application function (either you did not include a Server API directive or your application function for that step returned HTTP_NOACTION), the server performs the default action for that step.

Note: This is true for all steps except the Service step; the Service step does not have a default action.

The following list indicates the purpose of each step and defines the processing order.

Server Initialization
Performs initialization functions before any client requests are read.

PreExit
Performs processing after a request is read but before anything else is done.

If this step returns an indication that the request was processed (HTTP_OK), only the Data Filter, Log and PostExit steps, in the request process are performed.

Authentication
Decodes, verifies, and stores security tokens.

Name Translation
Translates the virtual path (from URL) to the physical path.

Authorization
Uses stored security tokens to check the physical path (protections, acls) and generates the WWW-Authenticate headers required for basic authentication. If you write your own application function to replace this step, you must generate these headers yourself.

Object Type
Locates the file system object that is indicated by the path.

Service
Satisfies the request (such as, send the file or run the CGI)

Data Filter
Gives write access to the outgoing data stream.

Log Allows transaction logging.

Error Allows customized responses to error conditions.

PostExit
Allows cleanup of resources that are allocated for request processing.

Server Termination
Allows cleanup processing when an orderly shutdown or restart occurs.

Application functions

Use the following function prototype syntax to write your own program functions for the defined request steps.

Each of your functions must fill in the return code parameter with a value that indicates the action that is taken.

- HTTP_NOACTION (value of 0) means no action that is taken.
- Otherwise, one of the valid HTTP return codes is expected, indicating that the application function handled the step. As a result, no other application functions are called to handle that step of this request.

The function prototypes for each request step show the format to use and explain the type of processing they can perform. You must give your functions unique names and can choose your own naming conventions. For ease of association, these names relate to the server's request processing steps.

Server Initialization

```
void
HTTPD_LINKAGE ServerInit(
     unsigned char *handle, unsigned long *major_version,
     unsigned long *minor_version, long *return_code);
```

This function is called once when your module is loaded during server initialization. This is your opportunity to perform initialization before any requests have been accepted. Although all server initialization functions are called, error return codes from this step cause the server to ignore all other functions that are configured in this program.

PreExit

```
void
HTTPD_LINKAGE PreExit(
          unsigned char *handle, long *return_code);
```

This function is called after the request is read, but before any processing has occurred.

All server-predefined functions are valid during this step.

Authentication

```
void
HTTPD_LINKAGE Authentication(
     unsigned char *handle, long *return_code);
```

This function allows user verification of the security tokens. This step is performed based on the authentication scheme.

Only HTTP_extract() and HTTPD_set() are valid during this step.

Name Translation

```
void
HTTPD_LINKAGE NameTrans(
     unsigned char *handle, long *return_code);
```

This function provides a mechanism for mapping URLs to objects.

Only HTTPD_extract() and HTTPD_set() are valid during this step.

Authorization

```
void
HTTPD_LINKAGE Authorization(
     unsigned char *handle, long *return_code);
```

This function verifies that the identified object may be returned to the client. If you are doing basic authentication, you must generate the required WWW-Authentication headers.

Only HTTPD_extract() and HTTPD_set() are valid functions during this step.

Object Type

```
void
HTTPD_LINKAGE ObjType(
     unsigned char *handle, long *return_code);
```

This step checks to see if the object exists and performs object typing.

Only HTTPD_extract() and HTTPD_set() are valid during this step.

Service

```
void
HTTPD_LINKAGE Service(
     unsigned char *handle, long *return_code);
```

This function satisfies the request, if not satisfied in the PreExit.

All server-predefined functions are valid during this step. Refer to the Enable directive in the *HTTP Server for AS/400 Webmaster's Guide*, GC41-5434 for information.

Data Filter

Filters data as a stream class, which means that each of its functions acts like a segment of pipe down which data flows. For this step, you must use three application functions:

```
void
HTTPD_LINKAGE open(
     unsigned char *handle, long *return_code);
```

This function performs any initialization (such as buffer allocation) that is required to process the data for this stream. An error return code causes this filter to abort.

```
void
HTTPD_LINKAGE write(
     unsigned char *handle, unsigned char *data,
     unsigned long *length, long *return_code);
```

This function processes the data and calls the server's write function with the new or changed data. The application must not attempt to free the buffer passed to it nor expect the server to free the buffer it receives.

```
void
HTTPD_LINKAGE close(
unsigned char *handle, long *return_code);
```

This function performs any cleanup (such as flushing and freeing the buffer) required to complete processing the data for this stream.

Log

```
void
HTTPD_LINKAGE Log(
     unsigned char *handle, long *return_code);
```

This function is called after each request is processed and the communication to the client is closed, regardless of the success or failure of the request processing. Only HTTPD_extract() and HTTPD_set() are valid during this step.

Error

```
void
HTTPD_LINKAGE Error(
     unsigned char *handle, long *return_code);
```

This function is called only when an error is encountered and provides an opportunity to customize the response.

PostExit

```
void
HTTPD_LINKAGE PostExit(
     unsigned char *handle, long *return_code);
```

This function is called, regardless of the success or failure of the request, so that you can clean up any resources that are allocated by your application to process the request.

Server Termination

```
void
HTTPD_LINKAGE ServerTerm(
     unsigned char *handle, long *return_code);
```

This function is processed when an orderly shutdown or restart of the server occurs. It allows you to clean up resources that are allocated during the Server Initialization step. Do not call any HTTP_* functions in this step (the results are unpredictable). If you have more than one Server API directive in your configuration file for Server Termination, they will all be called.

HTTP return codes and values

These return codes follow the HTTP specification that is published by the World Wide Web Consortium at URL: http://www.w3.org/pub/WWW/Protocols/. Your application functions should return one of these values.

Value	Return code
0	HTTP_NOACTION
100	HTTP_CONTINUE
101	HTTP_SWITCHING_PROTOCOLS
200	HTTP_OK
201	HTTP_CREATED
202	HTTP_ACCEPTED
203	HTTP_NON_AUTHORITATIVE
204	HTTP_NO_CONTENT
205	HTTP_RESET_CONTENT
206	HTTP_PARTIAL_CONTENT
300	HTTP_MULTIPLE_CHOICES
301	HTTP_MOVED_PERMANENTLY
302	HTTP_MOVED_TEMPORARILY
303	HTTP_SEE_OTHER
304	HTTP_NOT_MODIFIED
305	HTTP_USE_PROXY
400	HTTP_BAD_REQUEST
401	HTTP_UNAUTHORIZED
403	HTTP_FORBIDDEN
404	HTTP_NOT_FOUND
405	HTTP_METHOD_NOT_ALLOWED
406	HTTP_NOT_ACCEPTABLE
407	HTTP_PROXY_UNAUTHORIZED
408	HTTP_REQUEST_TIMEOUT
409	HTTP_CONFLICT
410	HTTP_GONE
411	HTTP_LENGTH_REQUIRED
412	HTTP_PRECONDITION_FAILED
413	HTTP_ENTITY_TOO_LARGE
414	HTTP_URI_TOO_LONG
415	HTTP_BAD_MEDIA_TYPE
500	HTTP_SERVER_ERROR
501	HTTP_NOT_IMPLEMENTED
502	HTTP_BAD_GATEWAY
503	HTTP_SERVICE_UNAVAILABLE
504	HTTP_GATEWAY_TIMEOUT
505	HTTP_BAD_VERSION

Predefined functions and macros

You can call the server's predefined functions and macros from your own application functions. You must use their predefined names and follow the format

that is described in this section. Note that the parameter descriptions use the letter″i″ to indicate input, the letter ″o″ to indicate output, and ″i/o″ to indicate that a parameter is both input and output.

Each of these functions returns one of the HTTPD return codes, depending on the success of the request.

HTTPD_authenticate()

Authenticates a user ID and password. Valid only in PreExit, Authenticate, and Authorization steps.

```
void
HTTPD_authenticate(
    unsigned char *handle,  /* i; handle */
    long *return_code);     /* o; return code */
```

HTTPD_attributes

Extracts the attributes of a file. Valid in all steps.

The name of the file and the buffer containing the attributes are in the default CCSID of the job.

```
void
HTTPD_attributes(
    unsigned char *handle,       /* i; handle (NULL right now) */
    unsigned char *name,         /* i; name of the file */
    unsigned long *name_length,  /* i; length of the name */
    unsigned char *value,        /* o; buffer containing attributes*/
    unsigned long *value_length, /* i/o;size of buffer/length of attributes*/
    long *return_code);          /* o; return code */
```

HTTPD_extract()

Extracts the value of a variable associated with this request. The valid variables you can use for the name parameter are the same as those used in the CGI. This function is valid in all steps, however not all variables are.

The CCSID of the name of the value to extract and the buffer in which to put the value depends upon the step and the CGI mode. For all steps except Service, these parameters are in the default CCSID of the job. For the Service step, the CGI mode determines the CCSID. For the %%MIXED%% CGI mode, these fields are in EBCIDIC CCSID 37. For all other CGI modes, these fields are in the default CCSID of the job.

```
void
HTTPD_extract(
    unsigned char *handle,        /* i; handle */
    unsigned char *name,          /* i; name of value to extract /*
    unsigned long *name_length,   /* i; length of the name */
    unsigned char *value,         /* o; buffer in which to put value */
    unsigned long *value_length,  /* i/o; buffer size/length of value */
             long *return_code);  /* o; return code */
```

If this function returns the HTTPD_BUFFER_TOO_SMALL return code, the buffer size you requested was not big enough for the extracted value. In this case, the function does not fill in the buffer but does update value_length with the buffer size you would need in order to successfully extract this value. Retry the extract with a buffer that is at least as big as the returned value_length.

Note: Server API programs gain access to information about a particular HTTP request by examining predefined variables. This information is obtained from the server using the HTTPD_extract() function. The PASSWORD variable used when basic authentication is requested.

This password is associated with a user that is defined in a validation list (*VLDL) or the password for a user profile on the AS/400 system.

IBM HTTP Server for AS/400 does not allow access to the PASSWORD variable if authorization is configured which uses AS/400 user profiles and passwords for authentication.

To prevent an application from obtaining an AS/400 user profile password, HTTPD_extract() is sensitive to the type of protect setups that are currently configured. If a protection setup is configured with a password file of %%SYSTEM%% (protection requiring AS/400 user profile password), HTTP_extract() for PASSWORD returns HTTP_PARAMETER_ERROR and sets the **value** parameter to *CONFLICT. Otherwise, HTTP_extract() returns the appropriate value.

HTTPD_reverse_translate()

Translates a file system path to a URL. Valid in all steps.

The name of the file system object and the buffer containing the URL are in the default CCSID of the job.

```
void
HTTPD_reverse_translate(
   unsigned char *handle,           /* i; handle (NULL right now) */
   unsigned char *name,             /* i; name of the file system object */
   unsigned long *name_length,      /* i; length of the name */
   unsigned char *value,            /* o; buffer which contains the URL */
   unsigned long *value_length,     /* i/o; size of buffer/length of URL */
            long *return_code);     /* o; return code */
```

HTTPD_translate()

Translates a URL to a file system path. Valid in all steps.

The CCSID for QUERY_STRING depends upon the step and the CGI mode. For all steps except Service, these parameters are in the default CCSID of the job. For the Service step, the CGI mode determines the CCSID. For the %%MIXED%% CGI mode, these fields are in EBCIDIC CCSID 37. For all other CGI modes, these fields are in the default CCSID of the job.

```
void
HTTPD_translate(
unsigned char *handle,           /* i; handle (NULL right now) */
unsigned char *name,             /* i; name of the URL */
unsigned long *name_length,      /* i; length of the name */
unsigned char *url_value,        /* o; buffer containing translated URL */
unsigned long *url_value_length,  /* i/o; buffer size/length of translated URL */
unsigned char *path_trans,       /* o; buffer containing PATH_TRANSLATED */
unsigned long *path_trans_length,/* i/o; size of buffer/length of PATH_TRANSLATED */
unsigned char *query_string,     /* o; buffer containing QUERY_STRING */
unsigned long *query_string_length,/* i/o; size of buffer/length of QUERY_STRING */
         long *return_code);     /* o; return code */
```

HTTPD_set()

Sets the value of a variable associated with this request. The valid variables you can use for the name parameter are the same as those are used by the CGI program.

Note that you can also create variables with this function. If any variables you create are prefixed by "HTTP_", they are sent as headers in the response, without the "HTTP_" prefix. For example, if you want to see a Location header, use HTTPD_set() with the variable name HTTP_LOCATION.

This function is valid in all steps, however not all variables are.

The CCSID of the name of the value to set and the buffer which contains the value depends upon the step and the CGI mode. For all steps except Service, these parameters are in the default CCSID of the job. For the Service step, if you are not setting a variable with the "HTTP"_prefix, then the default CCSID of the job is used. For setting variables prefixed by "HTTP_" in the Service step, the CGI output mode determines the CCSID to be used. For %%MIXED%% mode, both are in EBCIDIC 37. For other CGI modes, they are in the default CCISID of the job.

```
void
HTTPD_set(
    unsigned char *handle,        /* i; handle */
    unsigned char *name,          /* i; name of the value to set */
    unsigned long *name_length,   /* i; length of the name */
    unsigned char *value,         /* i; buffer containing the value */
    unsigned long *value_length,  /* i; length of value */
             long *return_code);  /* o; return code */
```

HTTPD_file()

Sends a file to satisfy this request. Valid only in PreExit, Service, NameTrans, Error, and DataFilter steps.

The name of the file to send is in the default CCSID of the job.

```
void
HTTPD_file(
    unsigned char *handle,  /* i; handle */
    unsigned char *name,    /* i; name of file to send */
    unsigned long *name_length,  /* i; length of the name */
             long *return_code); /* o; return code */
```

HTTPD_exec()

Runs a script to satisfy this request. Valid in PreExit, Service, NameTrans, and Error steps.

The name of the script to run is in the default CCSID of the job.

```
void
HTTPD_LINKAGE
HTTPD_exec(
    unsigned char *handle,       /* i; handle */
    unsigned char *name,         /* i; name of script to run */
    unsigned long *name_length,  /* i; length of the name */
             long *return_code); /* o; return code */
```

HTTPD_read()

This function reads the body of the client's request. It uses HTTPD_ extract for headers. This function is valid only in the PreExit, Service, and Data Filter steps.

For the Service step, the data CCSID is determined by the CGI mode. For %%MIXED%% the data is in the default job CCSID and any encoded sequences are the EBCIDIC representation of the ASCII character. For %%EBCIDIC%% and %%EBCIDIC_JCD%% modes, the data is in the default CCSID of the job (including the escape sequences). For %%BINARY%% mode, no conversion is performed. For all other steps, this is the default job CCSID.

```
void
HTTPD_read(
    unsigned char *handle,         /* i; handle */
    unsigned char *value,          /* i; buffer in which to place data */
    unsigned long *value_length,   /* i/o; buffer size/length of data */
             long *return_code);   /* o; return code */
```

HTTPD_write()

Writes the body of the response. Uses HTTPD_set and HTTPD_extract for headers. Valid in the PreExit, Service, NameTrans, Error, and Data Filter steps.

If you do not use HTTPD_set() to set the content type before the first time you call this function, the server assumes you are sending a CGI data stream.

You may need to use HTTPD_set() to set the CGI environment variable CONTENT_ENCODING to the appropriate code page for the content of your response before you send the data to the client.

For the Service step, the data to send is in the default job CCSID for %%MIXED%% and %%EBCIDIC%% CGI output modes unless overidden by a character CCSID tag on the content_type header. The data is %%BINARY%% mode, so it is assumed to be binary..

```
void
HTTPD_write(
    unsigned char *handle,         /* i; handle */
    unsigned char *value,          /* i; data to send */
    unsigned long *value_length,   /* i; length of the data */
              long *return_code);  /* o; return code */
```

HTTPD_log_error()

Writes a string to the server's error log.

The string passed to HTTPD_log_error must be EBCDIC (CCSID 37) data. The server converts the strinbg to the CCSID of the error log file.

```
void
HTTPD_LINKAGE
HTTPD_log_error(
    unsigned char *handle,        /* i; handle */
    unsigned char *value,         /* i; data to write */
    unsigned long *value_length,  /* i; length of the data */
              long *return_code);  /* o; return code */
```

HTTPD_log_trace()

Writes a string to the server's trace log.

The string passed to HTTPD_log_trace must be EBCDIC (CCSID 37) data.

```
void
HTTPD_LINKAGE
HTTPD_log_trace
    unsigned char *handle,        /* i; handle (NULL right now) */
    unsigned char *value,         /* i; data to write */
    unsigned long *value_length,  /* i; length of the data */
              long *return_code);  /* o; return code */
```

HTTPD_restart()

Restarts the server after all active requests have been processed. Valid only in ServerInit and ServerTerm steps.

```
void
HTTPD_restart(
    long *return_code);   /* o; return code */
```

HTTPD_proxy()

Makes a proxy request. Valid in PreExit and Service steps.

Note: This is a completion function; the response is complete after this function.

The name of the URL for the proxy request and the body of the request are in the default CCSID of the job.

```
void
HTTPD_LINKAGE
HTTPD_proxy(
    unsigned char *handle,        /* i; handle (NULL right now) */
    unsigned char *url_name,      /* i; url for the proxy request */
    unsigned long *name_length,   /* i; length of the url */
    unsigned char *request_body,  /* i; body of the request */
    unsigned long *body_length,   /* i; length of the body */
             long *return_code);  /* o; return code */
```

Note: Once an HTTPD_ function returns, it is safe for you to free any memory you passed with it.

Return codes

The server fills in the return code parameter with one of these values depending on the success of the request.

-1 HTTPD_UNSUPPORTED

The function is not supported.

0 HTTPD_SUCCESS

The function succeeded, and the output fields are valid.

1 HTTPD_FAILURE

The function failed.

2 HTTPD_INTERNAL_ERROR

The function encountered an internal error and cannot continue processing this request.

3 HTTPD_PARAMETER_ERROR

You may have added one or more incorrect parameters. For example, the variable you tried to extract is unknown.

4 HTTPD_STATE_CHECK

The function is not valid in this step.

5 HTTPD_READ_ONLY

Returned only by HTTP_set(). The application can not set the variable.

6 HTTPD_BUFFER_TOO_SMALL

Returned only by HTTPD_extract(). The provided buffer was too small.

7 HTTPD_AUTHENTICATION_FAILED

Returned only by HTTPD_authenticate(). Examine the HTTP_RESPONSE and HTTP_REASON variables for more information.

8 HTTPD_EOF

Returned only by HTTPD_read(). This return code indicates the end of the request body.

9 HTTPD_ABORT_REQUEST

The request has been aborted because the client has provided an entity that did not match the condition that is specified by the request.

10 HTTPD_REQUEST_SERVICED

Returned by HTTP_proxy. This return code indicates that the program that called the function completed the response for this request.

11 HTTPD_RESPONSE_ALREADY_COMPLETED

The function failed because the response for that request completed processing.

Server API configuration directives

Each step in the request process has a configuration directive that allows you to indicate which of your application functions you want called and run during that step. You can add these directives to your server's configuration file by manually editing it or by using the Server API Request Processing form in the server's Configuration and Administration forms.

Server API usage notes

Here are some notes to consider when using the Server API:

- When appropriate, you can indicate that you want your application function called for all URL requests or only for URL requests that match a specified mask.
- You can also have your Authentication functions called for every request or just for those with a type of Basic.
- The Server API directives, except for the Service and NameTrans directives, can be in any order in the configuration file. You do not need to include every Server API directive. If you do not have an application function for a particular step, just omit the corresponding directive.
- The Service and NameTrans directives work like the Exec directive and are dependent on its occurrence and placement relative to other mapping directives within the configuration file. This means that the server processes the Service, NameTrans, Map, Pass, Exec, Redirect, and Fail directives in their sequential order within the configuration file. When it successfully maps a URL to a file, it does not read or process any other of these directives.
- You can also have more than one configuration directive for a step. For example, you could include two NameTrans directives, each pointing to a different application function. When the server performs the name translation step, it will process your name translation functions in the order in which they appear in the configuration file.
- Multiple IP configuration (using a trailing IP address or template) is supported only for the Service and NameTrans directives.
- If the server fails to load a specific application function or you have a ServerInit directive that does not return an OK return code, no other application functions for that compiled Server API program will be called. Any processing specific to that program which was done up to this point is ignored. Other Server API programs that you include in these directives, and their application functions, are not affected.

Server API directives and syntax

Directive	Variable	Access path
ServerInit		/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name init_string
PreExit		/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name

Directive	Variable	Access path
Authentication	type	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name
NameTrans	/URL	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name IP_address_templete
Authorization	/URL	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name
ObjectType	/URL	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name
Service	/URL	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name* IP_address_template
Data Filter		/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name:function_name:function_name
Log	/URL	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name
Error	/URL	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name
PostExit	/URL	/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name
ServerTerm		/QSYS.LIB/MYLIB/MYGWAPI.SRVPGM:function_name

Server API directive variables

The variables in these directives have the following meanings:

type Used only with the Authentication directive. This variable determines if your application function is called. Valid values are:

Basic Application function is called only for basic authentication requests

***** Application function is called for all requests

/URL Determines for which URLs your application function is called. URL specifications in these directives are virtual (they do not include the protocol) but a slash (/) precedes them. For example, /www.ics.raleigh.ibm.com is correct, but http://www.ics.raleigh.ibm.com is not. Valid values are:

A specific URL
Application function is called only for that URL

URL templete
Application function is called only for URLs that match the template. You can specify a template as /URL*,/*, or *.

Note: An URL template is required with the Service directive if you want path translation to occur.

/path/file
The fully qualified file name of your compiled program

:function_name
The name you gave your application function within your program

In the DataFilter directive, you must supply the names of the open, write, and close functions.

The Service directive requires an asterisk (*) after the *:function_name*, if you want to have access to path information.

init_string
Optional on the ServerInit directive, this can contain any

arbitrary text you want to pass to your application function. Use HTTPD_extract() to extract the text from the INIT_STRING variable.

IP_address_template
Used only with the Service and NameTrans directives on servers that have more than one IP address. This variable determines if your application function is called only for requests that comes on a specific IP address or on a range of IP addresses.

Compatibility with other APIs

The Server API is compatible with other APIs, such as CGI. You can run your existing CGI programs on all the server's operating systems.

Porting CGI programs

Here are a few guidelines for porting CGI applications that are written in C to use the Server API:

1. Remove your main() entry point or rename it so you can build a service program.
2. Eliminate global variables or protect them with a mutual exclusion semaphore.
3. Change the following calls in your programs:
   - Change printf() header calls to HTTPD_set().
   - Change printf() data calls to HTTPD_write().
   - Change getenv() calls to HTTPD_extract(). Note, that this returns deallocated memory, so you must free the result.
4. Remember, that the server runs in a multi-threaded environment and your application functions must be threadsafe. If the functions are re-entrant, performance will not decrease.
5. Do not forget to set the Content-Type header if you are using HTTPD_write() to send data back to the client.
6. Check your code for memory leaks.
7. Think about your error paths. If you generate error messages yourself and send them back as HTML, you should return HTTPD_OK from your service functions.

Authentication and Authorization

Authentication is the verification of the security tokens that are associated with this request. Authorization is the process using the security tokens to determine if the requester has access to the resource. In the IBM HTTP Server, authentication is part of the authorization process; it occurs only when authorization is required.

If your Server API application provides its own authorization process, it will override the default server authorization and authentication. Therefore, if you have Authorization directives in your configuration file, the application functions associated with them must also handle any necessary authentication. The predefined HTTPD_authenticate() function is provided to assist you with this.

There are three ways you can provide for authentication in your authorization application functions:

- Write your own separate authorization and authentication application functions. In your configuration file, use both the Authorization and the Authentication directives to specify these functions. Be sure to include HTTPD_authenticate() in your authorization application function.

  When the Authorization step is run, it performs your authorization application function which, in turn, calls your authentication application function.
- Write your own authorization application function but have it call the default server authentication. In your configuration file, use the Authorization directive to specify your function. In this case, you will not need the Authenticate directive. Be sure to include HTTPD_authenticate() in your authorization application function.

  When the Authorization step is run, it performs your authorization application function which, in turn, calls the default server authentication.
- Write your own authorization application function and include all your authentication processing right into it. Do not use HTTPD_authenticate() in your authorization application function. In your configuration file, use the Authorization directive to specify your function. In this case, you will not need the Authentication directive.

  When the Authentication step is run, it performs your authorization application function and any authentication it included.

  If your Server API application does not provide its own authorization process, you can still provide customized authentication.

If your Server API application does not provide its own authorization process, you can still provide customized authentication. To do this, write your own authentication application function. In your configuration file, use the Authentication directives to specify your function. In this case, you do not need the Authorization directive.

Notes:

1. If you do not have any Authorization directives in your configuration file, or their specified application functions decline to handle the request, the server's default authorization will occur.
2. If you do have Authorization directives in your configuration file and their application functions include HTTPD_authenticate(), the server calls any authentication functions specified in the Authentication directives. If you do not have any Authentication directives defined, or their specified application functions decline to handle the request, the server's default authentication will occur.
3. If you do have Authorization directives in your configuration file but their application functions do not include HTTPD_authenticate(), no authentication functions will be called by the server. You must code your own authentication processing as part of your authorization application functions or make your own calls to other authentication modules.
4. The IBM HTTP Server automatically generates the challenge (by prompting the browser to return user ID and password) if you return 401 or 407 from your authorization exit. However, you must still configure a protection setup so that this will occur correctly.

Environment variables

You can use environment variables in the predefined functions HTTPD_extract() and HTTPD_set(). They represent values you can extract from a client request or

values you can set or create when processing a client request. For a list of environment variables you can use with Server API, see "Environment variables" on page 13.

Server API variables

ACCEPT_RANGES
Used to accept ranges other than bytes.

ALL_VARIABLES
All the CGI environment variables. For example:

```
ACCEPT_RANGES ON
AUTH_STRING
CLIENT_ADDR 9.67.84.3
```

CLIENT_ADDR
IP address for the client. For example:

```
9.67.193.2
```

CLIENT_HOST
Same as REMOTE_ADDR.

CLIENTMETHOD
HTTP method that is used in the request.

CLIENT_NAME
Host name of the machine making the request. For example:

```
kevin
```

CLIENT_PROTOCOL
Name and version of the protocol the client is using to make the request. For example:

```
HTTP
```

CONNECTIONS
Number of connections being served, or number of active requests. For example:

```
15
```

CONTENT_CHARSET
This is the character set of the response for text/*. For example:

```
US ENGLISH
```

CONTENT_TYPE_PARAMETERS
Other MIME attributes, but not the character set.

DOCUMENT_NAME
This is the name of the topmost document. If the HTML was generated by CGI, this contains the name of the CGI.

DOCUMENT_ROOT
As defined by pass rules.

DOCUMENT_URI
Same as DOCUMENT_URL.

DOCUMENT_URL
The Uniform Request Locator. For example:

```
http://www.anynet.com/~userk/main.htm
```

EXPIRES
Defines the expiration for documents that are stored in a proxy's cache.

ERROR_INFO
Specifies the error code to determine the error page. For example:

```
401
```

HTTP_COOKIE
Contains the cookie that is sent with this request to communicate state information. For example:

```
CustomerNumber=HJ68944
```

You can find details about cookies at URL:
http://www.netscape.com/newsref/std/cookie_spec.html.

HTTP_REASON
Sets the reason string in the HTTP response header.

HTTP_RESPONCE
Sets the response code in the HTTP response header.

INIT_STRING
The string specified on the ServerInit directive.

LAST_MODIFIED
For example:

```
12/25/97
```

LOCAL_VARIABLES
All the user-defined variables.

PASSWORD
For Basic authentication, contains the decoded password. For example:

```
password
```

Note: IBM HTTP Server for AS/400 does not allow access to the PASSWORD variable if authorization is configured which uses AS/400 user profiles and passwords for authentication.

To prevent an application from obtaining an AS/400 user profile password, HTTPD_extract() is sensitive to the type of protect setups that are currently configured. If a protection setup is configured with a password file of %%SYSTEM%% (protection requiring AS/400 user profile password), HTTP_extract() for PASSWORD returns HTTP_PARAMETER_ERROR and sets the **value** parameter to *CONFLICT. Otherwise, HTTP_extract() returns the appropriate value.

PATH Fully translated path.

PEAKCONNECTIONS
Defines the peak number of connections the server allows. For example:

```
45
```

PPATH
Partially translated path.

PROXY_ACCESS
Defines whether the request is a proxy request or not. For example:

```
NO
```

PROXY_CONTENT_TYPE
Content-Type header of the proxy request that is made through HTTPD_proxy. When information is sent with the method of POST, this

variable contains the type of data included. You can create your own content type in the server configuration file and map it to a viewer. For example:

```
application/x-www-form-urlencoded
```

PROXY_CONTENT_LENGTH
Content-Length header of the proxy request that is made through HTTPD_proxy. When information is sent with the method of POST, this variable contains the number of characters. Servers typically do not send an end-of-file flag when they forward the information by using stdin. If required, you can use the CONTENT_LENGTH value to determine the end of the input string. For example:

```
7034
```

PROXY_METHOD
Method for the request that is made through HTTPD_proxy.

QUERY_STRING_UNESCAPED
The search query sent by the client. This is undefined unless HTML was generated by a CGI program.

REQHDR
Defines a list of the headers that are sent by the client.

REQUEST_CONTENT_TYPE
When information is sent with the method of POST, this variable contains the type of data included. You can create your own content type in the server configuration file and map it to a viewer. For example:

```
application/x-www-form-urlencoded
```

RESPONSE_CONTENT_LENGTH
When information is sent with the method of POST, this variable contains the number of characters. Servers typically do not send an end-of-file flag when they forward the information by using stdin. If required, you can use the CONTENT_LENGTH value to determine the end of the input string. For example:

```
7034
```

SCRIPT_NAME
URL of the request.

SERVER_ADDR
Local IP address for the server.

SERVER_ROOT
Directory where the server program is installed.

SSI_DIR
The path of the current file relative to SSI_ROOT. If the current file is in SSI_ROOT, this value is "/".

SSI_FILE
The file name of the current file.

SSI_INCLUDE
The value that is used in the include command that retrieved this file. This is not defined for the topmost file.

SSI_PARENT
The path and file name of the includer, relative to SSI_ROOT.

SSI_ROOT
The path of the topmost file. All include requests must be in this directory or a child of this directory.

Example:

```
<!--#echo var=SSI_DIR -->
```

Note: You can use **echo** to display a value set by the set or global directives.

URI Same as DOCUMENT_URL

URL Same as DOCUMENT_URL

USERID
Same as REMOTE_USER.

USERNAME
Same as REMOTE_USER.

Note: All headers sent by the client (such as Set-Cookie) are prefixed by ″HTTP_″, and their values can be extracted. To access variables that are headers, prefix the variable name with ″HTTP_″. You can also create new variables using the HTTP_set() predefined function.

Chapter 8. Writing Java Servlets

Servlets are ordinary Java programs that use additional packages (and the associated classes and methods) found in the Java Servlet API. Servlets generate dynamic content for web pages and run on the server.

The server can load a servlet automatically when it starts, or when the first client makes a request for the services of the servlet. Once loaded, servlets stay running, waiting for additional client requests.

Servlets extend the capabilities of the web server by creating a framework for providing requests and or response services over the web. A client sends a request to the server. The server sends the request information to the servlet. The servlet then constructs a response that the server sends back to the client. Servlets run independently of protocol or platform.

Java servlets can offer a performance advantage. Because they run on the server, they don't require download time.This makes servlets faster than applications which run on the client's system and require download time. When you choose to use a servlet, you don't encounter difficulties with the application passing through a firewall. Java servlets also offer an additional performance advantage because they are multi-threaded.

Because it is a Java program, the servlet can use all the capabilities of the Java language in constructing the response. The servlet can also interact with outside resources, such as files, databases or other applications (also written in Java, or other languages), to construct the response.

The response to the client, therefore, can be a dynamic and unique response to the particular interaction, rather than an existing static HTML page.

Overview of servlets

Servlets perform a wide range of functions, such as:

- A servlet can create and return an entire HTML web page, with dynamic content based on the nature of the client request.
- A servlet can create just a portion of an HTML web page embedded in an existing static HTML web page.
- A servlet can communicate with other resources on the server, including databases, other Java applications, and applications written in other languages.
- A servlet can handle connections with multiple clients, accepting input from and broadcasting results to the multiple clients. A servlet could, for example, be a multi-player game server.
- You can develop a servlet that allows a client to upload an agent to the server where the server runs the servlet.

IBM's WebSphere™ Application Server provides the foundation for deploying and running your e-business™ applications. It provides a secure, reliable execution environment for Java servlets. You can build and test Java servlets for WebSphere,

all from a single integrated environment. The VisualAge® for Java servlet builder enables you to visually develop new Java servlets and run-test them automatically in a WebSphere test environment.

For more information about Java servlets and the Websphere Application Server, go to URL: **http://www.software.ibm.com/webservers/appserv/** .

Chapter 9. Using Server-Side Includes

This chapter discusses using server-side includes with IBM HTTP Server for AS/400.

Server-side includes allow you to insert information into CGI programs and HTML documents that the server sends to the client. This chapter describes the commands that are required to make server-side includes work in your CGI programs and HTML documents.

Considerations for using server-side includes

Before using server-side includes on your server, there are a few issues you should consider. One issue is performance. Performance can be significantly impacted when the server is processing files while sending them. Another issue is security. Allowing users to run commands can be a security risk. Care should be taken when deciding which directives you use server-side includes in and in which directories you use the exec command. You can minimize the security risk if you do not enable the exec command.

You should also note that you cannot refer to files recursively. For example, if you are running file sleepy.html and the program finds `<--!#include file="sleepy.html"-->` the server does not detect the error and the server loops until it abends. However, you can refer to files within files. For example, file sleepy.html refers to files smiley.html references dopey.html.

Preparing to use server-side includes

To use server-side includes, you must add the AddType directive to your configuration file. Two examples follow:

Examples:

```
AddType .shtml text/x-ssi-html 8 bit 1.0
AddType .htmls text/x-ssi-html 8 bit 1.0
```

Note: If you use file extensions other than .shtml or .htmls, you should check the AddType directive to see if that extension already exists. See the configuration file, appendix listing, or the MIME form for a list of existing AddType directives.

You can also use the imbeds directive to specify whether server-side includes are used in HTML documents, CGI programs, or both. Examples of this directive follow:

Examples:

```
imbeds value
imbeds on
```

Default: imbeds *off*

The server does not process your error files for imbeds, regardless of the file extensions or use of the imbeds directive.

Format for server-side includes

The current date, the size of a file, and the last change of a file are examples of the kind of information that can be sent to the client. There are commands that need to be included in the HTML document comments. The commands have the following format:

```
<!--#directive tag=value ...-->
<!--#directive tag="value" ...-->
```

The quotes around *value* are optional. However, quotes are required for imbedding spaces.

Directives for server-side includes

Use the **config** directive to control certain aspects of file processing. Valid tags include cmntmsg, errmsg, sizefmt, and timefmt.

cmntmsg — specifies the message appended to the beginning of text: Use this tag to specify the message that gets appended to the beginning of any text that follows a directive specification and comes before ″—>″.

Example:

```
<!--#config cmntmsg="[This a comment]"-->
<!--#echo var=" " extra text -->
```

Result: (Output from the echo) <!—This is comment extra text –>.

Default: [the following was extra in the directive].

errmsg — specify the message sent to the client: Use this tag to specify the message that is sent to the client if an error occurs when a file is being processed. The message is logged in the server's error log.

Example:

```
<!--#config errmsg="[An error occurred]" -->
```

Default: ″[an error occurred while processing this directive]″.

sizefmt — specify file size format: Use this tag to specify the format used when displaying the file size. In the following examples, bytes is the value used for a formatted number of bytes. abbrev displays the number of kilobytes or megabytes.

Example 1:

```
<!--#config sizefmt = bytes -->
<!--#fsize file=foo.html -->
```

Result: 1024

Example 2:

```
<!--#config sizefmt=abbrev -->
<!--#fsize file=foo.html -->
```

Result: 1K

Default: ″abbrev″

timefmt — specify date format: Use this tag to specify the format used when providing dates.

Example:

```
<!--#config timefmt="%T %D" -->
<!--#flastmod file=foo.html -->
```

Result: ″10/18/95 12:05:33″

Default: ″%a, %d %b-%Y %T %Z″

The following strftime() formats are valid with the timefmt tag:

Table 4. Conversion Specifiers Used by strftime()

Specifier	Meaning
%%	Replace with %
%a	Replace with the abbreviated weekday name.
%A	Replace with the full weekday name.
%b	Replace with the abbreviated month name.
%B	Replace with the full month name.
%c	Replace with the date and time.
%C	Replace with the century number (year divided by 100 and truncated)
%d	Replace with the day of the month (01-31)
%D	Insert the date as %m/%d/%y.
%e	Insert the month of the year as a decimal number (01-12). With C POSIX this field is a 2-characters long, right-justified, and blank filled.
%E[cCxyY]	If the alternative date and time format is not available, the %E descriptions are mapped to their unextended counterparts. For example, %E is mapped to %C.
%Ec	Replace with the alternative data and time representation.
%EC	Replace with the name of the base year in the alternative representation.
%Ex	Replace with the alternative data representation.
%EX	Replace with the alternative time representation.
%Ey	Replace with the offset from %EC (year only) in the alternative representation.
%EY	Replace with the full alternative year representation.
%h	Replace with the abbreviated month name. This is the same as %b.
%H	Replace with the hour (23-hour clock) as a decimal number (00-23).
%I	Replace with the hour (12-hour clock) as a decimal number (00-12).
%j	Replace with the day of the year (001-366).
%m	Replace with the month (01-12)
%M	Replace with the minute (00-59)
%n	Replace with a new line.

Table 4. Conversion Specifiers Used by strftime() (continued)

Specifier	Meaning
%O[deHlmMSUwWy]	If the alternative date and time format is not available, the %E descriptors are mapped to their unextended counterparts. For example, %Od is mapped to %d.
%Od	Replace with the day of the month, using the alternative numeric symbols. Fill as needed with leading zeros if there is any alternative symbol for zero, otherwise with leading spaces.
%Oe	Replace with the day of the month, using the alternative numeric symbols, filled as needed with leading spaces.
%OH	Replace with the hour (24 hour clock) using the alternative numeric symbols.
%OI	Replace with the hour (12 hour clock) using the alternative numeric symbols.
%Om	Replace with the month using the alternative numeric symbols.
%OM	Replace with the minutes using the alternative numeric symbols.
%OS	Replace with the seconds using the alternative numeric symbols.
%OU	Replace with the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the alternative numeric symbols.
%Ow	Replace with the weekday (Sunday=0) using the alternative numeric symbols.
%OW	Replace with the week number of the year (Monday as the first day of the week) using the alternative numeric symbols.
%Oy	Replace with the year (offset from %C) in the alternative representation and using the alternative numeric symbols.
%p	Replace with the local equivalent of AM or PM.
%r	Replace with the string equivalent to %I:%M:%S %p
%R	Replace with the time in 24 hour notation (%H:%M).
%S	Replace with seconds (00-61).
%t	Replace with a tab.
%T	Replace with a string equivalent to %H:%M:%S.
%u	Replace with a weekday as a decimal number (1 to 7), with a 1 representing Monday.
%U	Replace with the week number of the year (00-53) where Sunday is the first day of the week.
%V	Replace with the week number of the year (01-53) where Monday is the first day of the week.
%w	Replace with the weekday (0-6) where Sunday is 0.
%W	Replace with the week number of the year (00-53) where Monday is the first day of the week.
%x	Replace with the appropriate date representation.
%X′	Replace with the appropriate time representation.
%y	Replace with the year with the century.
%Y	Replace with the year with the current century.
%Z	Replace with the name of the time zone or no characters if the time zone is not known.

Note: The operating system configuration determines the full and abbreviated month names and years.

Use the **echo** directive to display the value for specified environment variables using the var tag. If a variable is not found, a **(None)** is displayed. In addition to the standard Common Gateway Interface (CGI) variables, you can also display the following environment variables:

DATE_GMT
The current date and time in Greenwich Mean Time. The config timefmt directive defines the formatting of this variable.

DATE_LOCAL
The current date and local time. The formatting of this variable is defined using the config timefmt directive.

DOCUMENT_NAME
This is the name of the topmost document. If the HTML was generated by CGI, this contains the name of the CGI.

DOCUMENT_URI
The full URL the client entered, without the query string.

LAST_MODIFIED
The current data and time that the current document was last changed. The config timefmt directive defines the formatting of this variable.

QUERY_STRING_UNESCAPED
The search query sent by the client. This is undefined unless the HTML was generated by a CGI program.

SSI_DIR
The path of the current file relative to SSI_ROOT. If the current file is in SSI_ROOT, this value is "/".

SSI_FILE
The file name of the current file.

SSI_INCLUDE
The value used in the include command that retrieved this file. This is not defined for the topmost file.

SSI_PARENT
The path and file name of the includer, relative to SSI_ROOT.

SSI_ROOT
The path of the topmost file. All include requests must be in this directory or a child of this directory.

Example:

```
<!--#echo var=SSI_DIR -->
```

Note: You can use **echo** to display a value set by the set or global directives.

You can use the **exec** directive to include the output of a CGI program. The Exec directive discards any HTTP headers CGI outputs except for:

content-type
determines whether to parse the body of the output for other Includes.

content-encoding
determines if EBCDIC to ASCII translation must be performed.

last-modified
replaces the current last modified header value if it is later.

You can use the **cgi** directive to specify the URL of a virtual path to a CGI program.

Example 1:

```
<!--#exec cgi="/cgi-bin/program/path_info?query_string" -->
```

Example 2:

```
<!--#exec cgi="&path;&cgiprog;&pathinfo;&querystring;" -->
```

This example shows the use of variables.

You can use the **flastmod** directive to display the last time and date the document changed. The config timefmt directive defines the formatting of this variable. The file and virtual tags can be used with this directive, and the meaning is the same as it is for the include directive.

Directive Formats:

```
<!--#flastmod file="/path/file" -->
<!--#flastmod virtual="/path/file" -->
```

Example:

```
<!--#flastmod file="FOO" extra text -->
```

Result: 12May96 <!— This is extra text –>

Use the **fsize** directive to display the size of the specified file. The formatting of this variable is defined using the config sizefmt directive. You can use the file and virtual tags with this directive, and the meaning is the same as it is for the include directive.

Examples:

```
<!--#fsize file="/path/file" -->
<!==#fsize virtual="/path/file" -->
```

Result: 1K

Use the **global** directive to define global variables that are echoed by this file, or any included files.

Example:

```
<!--#global var=VariableName value="Some Value" -->
```

If you want to refer to a parent document across the ″virtual″ boundary, you need to set a global variable DOCUMENT_URI. You must also refer to the global variable in the child document. The following is an example of the HTML coding you need to insert in the parent document:

Example:

```
<!--#global var="PARENT_URI" value=&DOCUMENT_URI; -->
```

The following is an example of the HTML coding you need to insert in the child document:

Example:

```
<!--#flastmod virtual=&PARENT_URI; -->
```

Use the **include** directive to include a document (the text from a document) in the output. You can use the file and virtual tags with the include directive.

file - specify file name: Use this tag to specify the name of a file.

For the flastmod, fsize, and include directives, the file tag is assumed to be relative to SSI_ROOT if preceded by a '/'. Otherwise, it is relative to SSI_DIR. The file specified must exist in SSI_ROOT or in one of its descendents.

Example:

```
<!--#include file="/path/file" -->
```

virtual - specify a document URL: Use this tag to specify the URL of a virtual path to a document.

For the flastmod, fsize, and include directives, the virtual tag is always passed through the server's mapping directives.

Example:

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

Use the **set** directive to set a variable that only this file can be echo later.

Example:

```
<!--#set var="Variable 2" value="AnotherValue" -->
```

Variables: Server-side includes also allow you to echo a variable already set. While defining a directive, you can also echo a string in the middle of ″value″. For example:

```
<!--#include file="&filename;" -->
```

Nothing is displayed if an unrecognized variable is found.

Server-side includes look for the variable, echoes where the variable is found, and proceeds with the function. You can have multiple variable references. When server-side includes encounter a variable reference inside a server-side include directive, it attempts to resolve it on the server side. The following example escapes the & so that server-side includes do not recognize it as a variable. In the second line of the example, the variable ″&index;″ is a server-side variable and is used to construct the variable name ″var1″. The variable ê is a client side variable, so the & is escaped to create the value ″:frêd″ or ″fred″ with a circumflex over the e.

```
<!--#set var="index" value="1"  -->
<!--#set var+"var&index;" value+"fr\&ecirc;d" -->
<!--#echo var="var1" -->
```

The following characters can be escaped. Escape variables must be preceded with a backslash (\).

\a	Alert (bell)
\b	Backslash
\f	Form feed (new page)
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\′	Single quote mark
\″	Double quote mark
\?	Question mark
\\	Backslash
\-	Hyphen
\.	Period
\&	Amphersand

Chapter 10. Troubleshooting your CGI programs

You can use the Work with Active Jobs (WRKACT JOB) command to check on the status of server jobs, as follows:

```
WRKACTJOB SBS(QHTTPSVR) JOB(server_instance)
```

When the server is not processing a request, the Work with Active Jobs display might be similar to Figure 1:

```
                          Work with Active Jobs                    AS400SYS
                                                                10/22/97  16:35:38
CPU %:     .4     Elapsed time:   00:02:01     Active jobs:   98

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function          Status

       DEFAULT      QTMHHTTP    BCH      .0  PGM-QZHBHTTP      TIMW
       DEFAULT      QTMHHTTP    BCI      .0                    TIMW
       DEFAULT      QTMHHTTP    BCI      .0                    TIMW
       DEFAULT      QTMHHTTP    BCI      .0                    TIMW

                                                                          Bottom
Parameters or command
===>
F3=Exit      F5=Refresh   F10=Restart statistics   F11=Display elapsed data
F12=Cancel   F23=More options   F24=More keys
```

Figure 1. WRKACTJOB SBS(QHTTPSVR) Job(DEFAULT)

To find out if server jobs have ended abnormally, check the spooled files that contain the job logs (QPJOBLOG) for the user profile QTMHHTTP.

The symptoms that are described in this section would be seen running a request to the AS/400 server at a browser.

Symptom

Connection abandoned, dropped, or no data sent.

Note: Different browser issues different messages when no data is returned to the browser. Abandoned, dropped or no data will be displayed at the browser.

Cause: The system has incorrectly formatted a CGI program that writes data to standard output. The data that is written to stdout may have one of the following problems:

- No data written to stdout
- No “Content-type”, “Location”, or “Status” line

- No new line character after HTTP response header
- No data after HTTP response header.

Solution: Write the data to stdout with "Content-type: " line with two new line characters ("\n") and the data to be returned to the client. For example:

```
Content-type: text/plain\n
  \n
  This data is returned to the client
```

Cause: CGI program caused an exception message that was not handled by the CGI program.

Solution: Look at the active server job logs in Figure 1 on page 139. If the system does not indicate a message in the joblog for the active server jobs, do a WRKSPLF QTMHHTTP. Check for server jobs that ended when the system ran the CGI program. Change the program to monitor for the unhandled message.

Cause: The program being called does not exist in the library.

Solution: Check the library for the correct name.

Cause: There is a bug in your user-created CGI program.

Solution: You need to set up a scaffolding environment to debug the CGI application prior to integration with server:

1. Issue the command ENDTCPSVR *HTTP HTTPSVR(*server_instance*)
2. Issue the command STRTCPSVR *HTTP HTTPSVR(*server_instance* '-minat 1 -maxat 1')

   Note: You also may need to change script_timeout and output_timeout to be larger. If you are stepping through your code, it may take too long and script_timeout or output_timeout may expire. This causes the server to terminate the job you are debugging.

   Ending and starting the server ensures that only one worker job is running.

   a. Issue the command WRKACTJOB JOB(*server_instance*)

      Three active jobs are displayed for this server instance. The first job in the list is always the server instance server job (Function PGM-QZHBHTTP). The second and third jobs in the list are the CGI program jobs. One is for single thread-capable CGI programs, and the other is for multithread-capable CGI programs (Java).

      Select option 10 to display the job log.

      If your CGI program is single thread capable (not written in Java language), message HTP2001 will be in the job log. If your CGI program is multithread capable (Java), message HTP2002 will be in the job log.

      Record the *Number:*, *User:*, and *Job:* values for your CGI program job.

      Press F12.

      Issue the command STRSRVJOB <Number/User/Job>.

   b. For the user CGI program, issue the command STRDBG <*usercgilib/cgipgm*>

      If the program accesses a database file on the AS/400, you must specify UPDPROD(*YES). See the help for the STRDBG command.

Note: You will probably need additional authority to troubleshoot the CGI program. For example, you will probably need authority to the QTMHHTTP user profile.

c. Set breakpoints in the program.

d. On the browser, issue a URL that would run the CGI program.

The *Pass* and *Exec* directives that request the document and run a CGI program must be in WRKHTTPCFG.

e. After the system issues an HTTP request on the browser, return to the AS/400 session that ran STRSRVJOB. It should have stopped at a program breakpoint.

Ending and starting the server ensures that only one worker thread is running.

3. When finished with debug, reset the server values:
   a. Issue the command ENDDBG
   b. Issue the command ENDSRVJOB
   c. Issue the command WRKACTJOB SBS(QHTTPSVR) JOB*server_instance*
   d. Issue the command STRTCPSVR *HTTP HTTPSVR(*server_instance*

Symptom

The system is not converting or handling special characters as expected.

Cause: The browser inserts special characters using escape sequences which requires special handling by the CGI program.

Solution: Browsers create escape sequences (ISO 8859) for special characters (for example, : . , ! @ # $ % *, and so on.) These characters come into standard input or into the QUERY_STRING environment variable in the form "%xx", where "xx" is the two characters representing the ASCII hexadecimal value. (For example, a comma comes in as "%2C". For CGI input mode %%MIXED%%, these three characters "%xx" are converted to EBCDIC, but the values of "xx" are not changed to the corresponding EBCDIC code points.

There are two approaches to handling escape sequences:

1. Convert the EBCDIC representation of the ASCII escape sequence to an EBCDIC escape sequence or use CGI input mode %%EBCDIC%%. This is necessary because the *QtmhCvtDB* API assumes that escape sequences represent EBCDIC code points, and the API converts them to the corresponding EBCDIC character. For example, %2C, which represents an ASCII comma, is converted to EBCDIC X'2C', which is not an EBCDIC comma.
2. Convert the EBCDIC representation of the ASCII escape sequence to the EBCDIC equivalent character.

The following approach outlined in the first conversion technique listed above:

Note: The hex representation of the %2C from the browser was 0x253243. When this escape sequence is converted to EBCDIC, it ends up as 0x6CF2C3.

1. Convert the "xx" in "%xx" to the corresponding EBCDIC character. In this case 0xF2C3 is converted to 0x2C.
2. For the first approach, convert the EBCDIC character to the two-byte form. Then you can reinsert the two bytes back into the input stream in the same

place they originally appeared. The 0x6B would be converted to 0xF6C2, and the resultant escape sequence would be 0x6CF6C2.

For the second approach, leave the data in its EBCDIC form and replace the original escape sequence (three characters) with the single character. In this case, replace 0x6CF2C3 with 0x6B.

Note: The CGI program should preserve an escape sequence that represents the character "%".

3. Call *QtmhCvtDB* to convert the input stream.

Note: 7-bit ASCII CCSID 367 is standard on browsers.

Symptom

Error 403: Forbidden - Path not valid for this server.

Whenever a "Forbidden - Path not valid for this server" occurs when running a CGI program, the configuration directives have not been specified correctly.

Cause when a CGI program is requested: When a CGI program is requested, a Pass directive appears before an Exec directive. For example:

```
Pass  /qsys.lib/htmlcgi.lib/*
Exec  /qsys.lib/htmlcgi.lib/*
```

In this example any programs in library htmlcgi will not run because the *Pass* occurred before the *Exec.* Once a *Pass* condition is true, then the server does not go further.

Solution when a CGI program is requested: The best way to avoid this problem is to use one of the following:

1. Use *Exec* and *Pass* directives with mapping:

   ```
   Pass /doc/* qsys.lib/html.lib/*
   Exec /cgi-bin/* qsys.lib/html.lib/*
   ```

2. Put the CGI programs in a separate library:

   ```
   Exec  /qsys.lib/htmlcgi.lib/*
   Pass  /qsys.lib/htmldoc.lib/html.file/*
   ```

Cause when a document is requested: When a document is requested, an Exec directive appears before a Pass directive. For example:

```
Exec  /qsys.lib/html.lib/*
Pass  /qsys.lib/html.lib/*
```

In this example any documents in library html will not be found because the *Exec* occurred before the *Pass.*

Solution when a document is requested: Change the order of the directives to correct the problem. For example:

```
Pass  /qsys.lib/html.lib/*
Exec  /qsys.lib/html.lib/*
```

Note: Since the value mapped `/qsys.lib/html.lib/` is the same for both *Pass* and *Exec*, the combination above would correct a problem with using an incorrect directive. It also would leave a directive in the file that could never be used.

The best way to avoid this problem is to use one of the following:

1. Use *Exec* and *Pass* directives with mapping:

   ```
   Pass /doc/* qsys.lib/html.lib/*
   Exec /cgi-bin/* qsys.lib/html.lib/*
   ```

2. Put the CGI programs in a separate library:

   ```
   Exec  /qsys.lib/htmlcgi.lib/*
   Pass  /qsys.lib/htmldoc.lib/html.file/*
   ```

Symptom

Error 500: Bad script request -- script '/qsys.lib/qsyscgi.lib/progname.pgm' not found or not executable

Cause: Incorrect match of *Exec* rule.

This message can appear for the following reasons:

- The script does not exist.
- There is a problem with the script, for example, a send error or function check.
- The user QTMHHTTP does not have authority to run this program.

Solution: See Description of "Forbidden - Path not valid for this server".

Symptom

A browser request that runs a CGI program runs longer than expected. The browser keeps waiting for a response.

Cause: The CGI application that was running has taken a function check.

Solution: Look at the QSYSOPR message queue for a message that requires a reply sent from the CGI program that was running. Note the statement where the program is failing. Use the procedure described under "Symptom: Error 500".

Chapter 11. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator
3605 Highway 52 N
Rochester, MN 55901-7829
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

This product includes computer software created and made available by CERN. This acknowledgment shall be mentioned in full in any product which includes the CERN computer software included herein or parts thereof.

Programming Interface Information

This publication is intended to help you to write external programs to communicate and interact with the IBM HTTP Server for AS/400. This publication documents General-Use Programming Interface and Associated Guidance Information provided by IBM HTTP Server for AS/400.

General-Use programming interfaces allow the customer to write programs that obtain the services of IBM HTTP Server for AS/400.

Trademarks

The following terms are trademarks of IBM Corporation in the United States or other countries or both.

AS/400
DB2
e-business
IBM
Integrated Language Environment
Net.Data
VisualAge
WebSphere

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Other company, product, and service names may be trademarks or service marks of others.

Readers' Comments — We'd Like to Hear from You

AS/400e
HTTP Server for AS/400 Web Programming Guide

Publication No. GC41-5435-04

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
GC41-5435-04

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
ATTN DEPT 542 IDCLERK
3605 HWY 52 N
ROCHESTER MN 55901-7829

Fold and Tape | **Please do not staple** | Fold and Tape

GC41-5435-04

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC41-5435-04

Spine information:

AS/400e

Web Programming Guide V4R5