Chapter 13

WEB PROGRAMMING INTERFACES

CGI APPLICATION PROGRAMMING INTERFACE (API) 587

QtmhGetEnv – Get Value of Environment Variable 588

QtmhPutEnv – Put Environment Variable 590

Environment Variables 592

QtmhRdStin – Read from Standard-input 594

QtmhWrStout – Write to Standard-output 595

QtmhCvtDB – Convert to Database Structure 596

URL ENCODING 600

atoll() – C Runtime Library Function 601

Web development with a variety of software technologies has become pervasive. Many programming languages have integrated functions that support the development of Web-based applications. This class of applications uses the Common Gateway Interface or “CGI” protocol to communicate through a Web server, such as Apache, with a Web browser. The RPG language has no built-in functions that support CGI/Web programming. CGI originated on the UNIX operating systems and is based on the C language I/O processing. RPG does not directly support his type of I/O processing. It does, however, have access to a set of APIs for CGI/Web development that make CGI programming possible.

CGI Application Programming Interface (API)

The CGI APIs in Table 13.1 provide a method of communication between with a Web browser and CGI RPG programs through the Web server. The prototypes for these APIs are not included with OS/400, but they are included in this chapter.

Table 13.1: CGI Application Programming Interfaces

API Name	Description
QtmhGetEnv	Get a value from the CGI environment.
QtmhPutEnv	Change a value in the CGI environment.
QtmhRdStin	Read a string from the standard-input device (i.e., read data sent to a CGI program from a HTML form).
QtmhWrStout	Write a string to the standard-output device (i.e., write to the Web browser).
QtmhCvtDB	Convert URL-encoded string to an externally described data structure.
QzhbCgiParse	Parse the data from an HTML form.
QzhbCgiUtils	“Utilities” to create a full HTTP response.
QzhbCgiSendState	Send or save CGI state information.
QzhbCgiRecvState	Revieve CGI state information.

These APIs are referred to as *bindable* APIs. That is, they are procedure calls, not program calls. Consequently, in RPG IV the CALLB or CALLP operation codes may be used to call CGI APIs. In addition, the API names are case-sensitive.

These APIs exist in a service program (*SRVPGM) on the system. The service program name is QZHBCGI, and it is located in the QHTTPSVR library.

The CALLB operation code can be used to call any CGI API. When doing so, traditional CALLB/PARM syntax is used; the values for the parameters are specified on the result field of the PARM operation.

While CALLB/PARM syntax is supported, it is not widely used nor is it encouraged for bindable API calls. Instead, the CALLP (call with prototype) syntax is preferred.

QtmhGetEnv – Get Value of Environment Variable

The QtmhGetEnv API retrieves a value from the *environment*. The environment is a carryover from the UNIX operating system environment to the HTTP Web server world. It contains information about the Web server, the client's Web browser, and the Web session. In addition, text entered into an HTML form may be transmitted to the CGI program by retrieving it from the environment.

Data is stored in the environment by assigning it to identifiers referred to as *variables*. An environment variable‘s data is retrieved from the environment.

All environment values are character in format; therefore, if a numeric value is returned it is returned as text, such as "1234". If the numeric value needs to be used as an actual number, the character string must be converted to numeric. This can be accomplished by calling the %INT or %DEC built-in function or if the version of RPG being used does not support this conversion, the C runtime library functions atoll() function may be used for integer values.

The parameters for the QtmhGetEnv API are illustrated in Table 13.2.

Table 13.2: QtmhGetEnv API Parameters

Parameter	Definition	Description
Return value (OUTPUT)	Char(*)	A variable that receives the data from the QtmhGetEnv API. This field should be large enough to receive the environment variable's value.
Return value length (INPUT)	10i0	A 4-byte integer that indicates the length of the field specified on the return value (parameter 1).
Bytes available (OUTPUT)	10i0	A 4-byte integer that receives the length of the environment variable data. This value, returned by the API, indicates the actual length of the environment value.
Environment variable name (INPUT)	Char(*)	A variable that contains the name of the environment variable being retrieved.
Environment variable length (INPUT)	10i0	A 4-byte integer that indicates the length of the environment variable name (parameter 4). This length represents the length of the environment variable name not the length of the field used for the environment variable. For example, when "HTTP_COOKIE" is specified for the environment variable name, the length parameter must be 11.
Standard API structure	Char(*)	A data structure that is passed by reference. It must be the standard API error structure.

To call QtmhGetEnv with CALLP, a prototype is required. The RPG IV source code in Example 13.1 can be used as the prototype for QtmhGetEnv.

Example 13.1: Prototype for QtmhGetEnv

```
.....DName+++++++++++EUDSFrom+++To/Len+TDc.Functions++++++++++++++++++++++++++++
     D QtmhGetEnv      PR                  ExtProc('QtmhGetEnv')
     D  szEnvVarRtnVal            65535A   OPTIONS(*VARSIZE)
     D  nEnvVarRtnLen                10I Ø CONST
     D  szEnvVarName                256A   CONST OPTIONS(*VARSIZE)
     D  nEnvVarNameLen               10I Ø CONST
     D  apiError                           LikeDS(QUSEC)
```

The RPG IV source code in Example 13.2 illustrates calling the QtmhGetEnv API, using a prototype.

Example 13.2: Calling the QtmhGetEnv API using a prototype

```
.....DName+++++++++++EUDSFrom+++To/Len+TDc.Functions++++++++++++++++++++++++++++
     D rtnVal          S           4Ø96A
     D method          S             1ØA
     D rtnBufLen       S             1ØI Ø
     D envVar          S             64A   Inz('REQUEST_METHOD')
     D                                     VARYING
     D envVarLen       S             1ØI Ø
     D nBytesRtn       S             1ØI Ø
     C                   Eval      QUSEC = *ALLX'ØØ'
     C                   Eval      rtnBufen = %size(rtnVal)-1
     C                   Eval      envVarLen = %len(%TrimR(envVar))
     C                   CallP     QtmhGetEnv(rtnVal : rtnBufLen :
     C                                    nBytesRtn  : envVar :
     C                                    envVarlen  : qusec )
     C                   if        nBytesRtn > Ø
     C                   Eval      method = %subst(rtnVal:1:nBytesRtn)
     C                   if        Method = 'POST'
     C                   callp     ReadStdInput()
     C                   elseif    Method = 'GET'
     C                   callp     ReadFromEnv()
     C                   endif
     C                   endif
```

In this example, the QtmhGetEnv API is used to retrieve the method used to send data from the browser to the Web server. If Method=POST, then standard-input needs to be used to retrieve the data from the Web server. If Method=GET then the environment QUERY_STRING must be retrieved to read the data.

QtmhPutEnv – Put Environment Variable

The QtmhPutEnv API allows an environment variable's value to be set. A new or existing value can be set using this API. This API is used to create or change an environment variable value. Normally it is used to pass data between application programs that are compatible with RPG's program-to-program call/parm structure. For example, if a Java application evokes a CGI RPG IV program, that RPG IV program could set environment variables that are subsequently used by the Java application.

An environment variable is set by assigning a value to it using the following format:

environment variable = value

For example: **ItemNo=12345**

This assigns the value "12345" to an environment variable named "ItemNo". If "ItemNo" already exists, its value is replaced. If it does not exist, it is created.

Table 13.3.: QtmhPutEnv API Parameters

Parameter	Definition	Description
Environment variable (INPUT)	Char(*)	A variable or literal that contains the name and value of an environment variable. The format is:*variable-name=value,* where *variable-name* is the name of the environment variable being set, and *value* is its value.
Length of the environment variable (INPUT)	Int(4) 10i0	A 4-byte integer that indicates the length of the text specified for the first parameter.
Standard API error structure	Char(*)	A data structure that is passed by reference. It should be a standard OS/400 API error data structure.

The CALLB operation code can be used to call QtmhPutEnv. When doing so, traditional CALLB/PARM syntax is used, specifying the parameters in the result field of the PARM operation. In addition, the CALLP (call with prototype) can be used to call QtmhPutEnv if a prototype is created for the API. The RPG source code in Example 13.3 may be used as a prototype for QtmhPutEnv.

Example 13.3: Example prototype for QtmhPutEnv

```
.....DName+++++++++++EUDS.......Length+TDc.Functions+++++++++++++++++++++
     D QtmhPutEnv      PR                  ExtProc('QtmhPutEnv')
     D  EnvVarValue                      *   VALUE OPTIONS(*STRING)
     D  EnvValueLen                    10I Ø CONST
     D  QUSEC                              OPTIONS(*VARSIZE) Like(QUSEC)
```

The RPG IV statements in Example 13.4 illustrate the call syntax for this API when the prototype is used.

Example 13.4: Call sytax for QtmhPutEnv with prototyping

```
.....DName+++++++++++EUDS.......Length+TDc.Functions+++++++++++++++++++++++++
      /COPY QSYSINC/QRPGLESRC,QUSEC
     D EnvVar          S            255A   VARYING
     D apiError        DS                  LikeDS(QUSEC) Inz(*ALLX'ØØ')

.....ClrnØ1..............OpCode(ex)Extended-factor2+++++++++++++++++++++++++++
     C                   Eval      EnvVar = 'CUSTNO=12345'
     C                   CallP     qtmhPutEnv(envVar: %Len(EnvVar) :
     C                                        apiError)
```

Several built-in environment variables are provided with the HTTP server. These environment variables can be changed with the QtmhPutEnv API and retrieved using the QtmhGetEnv API.

Environment Variables

While there are many environment variables, four are frequently used, including:

REQUEST_METHOD: Returns the method used to send data from an HTML form to the CGI program. The possible return values are GET and POST.

- If the REQUEST_METHOD equals POST (the recommended method) the data sent to the RPG IV program from the HTML form is available by reading it into the program from the standard-input device. To read from standard-input, the QtmhRdStin API is used.

- If the REQUEST_METHOD equals GET (used primarily by older Web sites), the data sent to the RPG IV program from the HTML form is available through the environment. To read information sent to the CGI program by the GET method, another environment variable, QUERY_STRING, is retrieved.

CONTENT_LENGTH: Returns the length of the data sent from an HTML form to the CGI program when REQUEST_METHOD=POST. Like all environment variables, the data returned for CONTENT_LENGTH is in character format. To utilize the length as a numeric value, it must be converted to numeric using one of the available techniques. Calling the C language runtime library function atoll() is one such technique.

- QUERY_STRING – Returns the data sent from an HTML form to the CGI program when REQUEST_METHOD equals GET. Normally, the length of the QUERY_STRING value is determined first by retrieving the value for the CONTENT_LENGTH environment variable. However, although the CONTENT_LENGTH is often set to the length of the QUERY_STRING data, it is not guaranteed to be set properly when REQUEST_METHOD=GET.

- HTTP_USER_AGENT – Returns the name of the Web browser being used by the client. This is useful when generating HTML and customization, as necessary based on the kind of Web browser being used.

Table 13.4 summarizes several environment variables used by CGI programs.

Table 13.4: Common Environment Variables

Environment Variable	Contains
REQUEST_METHOD	The value of the METHOD keyword in the HTML form. This indicates whether the URL-encoded string (form data) is being sent via the environment (when GET is returned) or via standard-input (when POST is returned).
CONTENT_LENGTH	The length of the URL-encoded string created from an HTML form. This is the length of the data sent to the CGI program as a result of METHOD="GET" being specified on an HTML form, when the SUBMIT button is pressed. All environment variables are returned as character values, so this value will need to be converted to decimal in order to us it properly. Tip: Use the C language runtime function atoi() to convert the length to decimal.
QUERY_STRING	The data sent to the CGI program from an HTML form when the METHOD="GET" is specified. This is the URL-encoded string created by the Web browser and set to the server. The string is coded in the standard URL format. Spaces are converted into "+" and special characters are converted into hex in the format "%xx". The CGI program must decode this information. APIs that are useful for decoding a URL-encoded string includes QzhbCgiParse and QtmhCvtDB.
REMOTE_ADDR	The IP address of the Web browser making the CGI request.
HTTP_USER_AGENT	The name of the Web browser client. If the Web client is the Netscape browser, HTTP_USER_AGENT may return something like this: Netscape Navigator dll /v3.0 If the Web browser is Microsoft Internet Explorer, HTTP_USER_AGENT may return something like this: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
HTTP_COOKIE	The cookies from the client's PC for the Web site. All cookies are returned in the form: name=value; name2=value2; name3=value3;…For example, if the cookie named CUSTNO is set to 1235, it is returned as follows: CUSTNO=12345;

QtmhRdStin – Read from Standard-input

The QtmhRdStin API retrieves the data sent to the CGI program through the standard-input device. The standard-input device or *stdin* is a carryover from the UNIX environment. The use of stdin in RPG IV is similar to reading data from a database file; however, instead of using the READ operation to read from the file, the QtmhRdStin API is used.

Table 13.5: QtmhRdStin API Parameters

Parameter	Definition	Description
Variable to receive the data from stdin	Char(*) - Output	A variable that receives the data read from the stdin device. This variable should be large enough to receive all data sent to the CGI program in one call to the API.
Length of the field specified on the first parameter	Int(4) 10i0 - Input	A 4-byte integer that indicates length of the variable specified on the first parameter. This is the maximum number of bytes that the read to standard-input can return. If the bytes returned (third parameter) are less than the value specified for this parameter, end-of-data has been reached.
Length of the data returned by the read to stdin	Int(4) 10i0 - Output	A 4-byte integer that indicates length of the data returned into the first parameter. This is the length of the data retrieved.
Standard API error structure	Char(*)	A data structure that is passed by reference. It should be a standard OS/400 API error data structure.

The prototype for QtmhRdStin is shown in Example 13.5.

Example 13.5: Prototype for QtmhRdStin

```
.....DName+++++++++++EUDS.......Length+TDc.Functions++++++++++++++++++++++++++
     D QtmhRdStin      PR                  ExtProc('QtmhRdStin')
     D  szRtnBuffer               65535A   OPTIONS(*VARSIZE)
     D  nBufLen                      10I 0 CONST
     D  nRtnLen                      10I 0
     D  QUSEC                              Like(QUSEC)
```

The source code in Example 13.6 illustrates the calling convention for QtmhRdStin using the standard CALLB/PARM syntax.

Example 13.6: Calling QtmhRdStin

```
.....DName+++++++++++EUDS.......Length+TDc.Functions+++++++++++++++
      /COPY QSYSINC/QRPGLESRC,QUSEC
     D RtnBuffer       S           4Ø96A   INZ
     D RtnLen          S             1ØI Ø
     D nRtnBufLen      S             1ØI Ø
     D apiError        DS                  LikeDS(QUSEC) Inz(*ALLX'ØØ')

.....ClrnØ1Factor1+++++++OpCode(ex)Factor2+++++++Result++++++++++++++++++
     C                   CallP     QtmhRdStin(rtnBuffer: %size(rtnBuffer) :
     C                                        rtnLen : apiError)
```

Logically, in order to use QtmhRdStin to read data sent to the CGI program, the value for the REQUEST_METHOD environment variable should be retrieved first. If REQUEST_METHOD equals POST, QtmhRdStin can be used to read the form data. If REQUEST_METHOD equals GET, then QtmhGetEnv can be used to read the form data.

Before calling QtmhRdStin, however, the length of the form data needs to be determined. The CONTENT_LENGTH environment variable can be used to determine the length of the data about to be retrieved. If the length of the variable being used to receive the data from standard input is less than the value returned to the CONTENT_LENGTH environment variable, not all data is returned by the API.

The data returned by a call to the QtmhRdStin API is in URL-encoded format. URL-encoded format is the format created by the Web browse when it sends data to the CGI program. It is an escaped format that may include hexadecimal data. Depending on the configuration of the Web server, this hexadecimal may be EBCDIC characters that represent ASCII characters when they are converted to character. The API 'cvtch' (convert from hex to character) may be used to convert the hexadecimal variable from their hexadecimal notation to single-character notation. Once converted, if the resulting character is an ASCII character, it must be converted to the CCSID of the job using the 'iconv' API.

QtmhWrStout – Write to Standard-Output

The QtmhWrStout API sends data to the Web browser through the *standard-output device.* The standard-output device or *stdout* is a carry over from the UNIX environment. The use of stdout in RPG IV is similar to writing data to a database file; however the QtmhWrStout API is used to write data instead of the WRITE operation code.

The QtmhWrStout API is the only way to send data such as HTML to a Web browser from an RPG IV program. Without this API, a routine written in a secondary language, such as C, would be required.

Table 13.6: QtmhWrStout API Parameters

Parameter	Definition	Description
Text to send to Web browser	Char(*) - Input	A variable that contains the text (usually HTML) that is sent to the Web browser.
Length of the text specified on the first parameter.	Int(4) 10i0 - Input	A 4-byte integer that indicates length of the text contained in the variable specified on the first parameter. This is the number of bytes (text length) of the HTML source stored in the field specified on the first parameter. This value is often less than the length of the field. By specifying the actually length of the HTML source, instead of the field length, performance can be improved.
Standard API structure	Char(*)	A data structure that is passed by reference. It should be a standard OS/400 API error data structure.

The prototype for QtmhWrStout is illustrated in Example 13.7.

Example 13.7: Prototype for QtmhWrStout

```
.....DName+++++++++++EUDS.......Length+TDc.Functions++++++++++++++++
     D QtmhWrStout     PR                  ExtProc('QtmhWrStout')
     D  szHtml                    65535A   Const OPTIONS(*VARSIZE)
     D  nBufLen                      10I 0 CONST
     D  QUSEC                              Like(QUSEC)
```

Frequent calls to QtmhWrStout can slow down the Web page being created. If possible, buffer the output and send it with as few calls to QtmhWrStout as possible. This does not mean that only one call to QtmhWrStout is required for good performance; it simply means if you send data one character at a time via QtmhWrStout expect that process to take more time than buffering it up and sending it all at once.

QtmhCvtDB – Convert to Database Structure

The QtmhCvtDB API provides an interface to convert URL-encoded strings into a data structure format. Normally, the data structure has been formatted based on an externally described database file.

The URL-encoded string that is received into the CGI program by the QtmhRdStin or QtmhGetEnv APIs as a character string is converted by QtmhCvtDB to the format specified for the fields of the data structure using a format file.

QtmhCvtDB works by matching the field names in the URL-encoded string to the fields in the database format file. It then converts the text assigned to the fields in the URL-encoded string into the data type and length required by the fields of the format file. The converted data is then copied to the target field in the data structure. It is strongly suggested that the field names on the HTML form be the same as those in the database format file. If they are not, QtmhCvtDb will fail.

While not all data types are supported by QtmhCvtDb, most generally used data types are supported. The following DDS data types are supported for conversion by QthmCvtDb:

- **A** Alphanumeric (fixed length only, VARLEN fields are not supported)
- **P** Packed Decimal
- **S** Zoned Decimal
- **F** Floating Point
- **T** Time
- **L** Date (data type "D" in RPG IV)
- **Z** Timestamp
- **B** Binary (data type "I" in RPG IV)
- **O** DBCS (data type "G" in RPG IV)

The following DDS data types are not supported by the QtmhCvtDB API:

- **H** Hexadecimal
- **G** Graphic
- **J** DBCS
- **E** DBCS

In addition to the above, the following SQL data type is not supported:

- **BLOB** Binary-large objects

As an example of how QtmhCvtDb converts data, assume an end-user enters the number 24.95 into a field named AMOUNT on an HTML form. All data on HTML forms is text, even if the field is considered numeric. That is, the field's value is always sent to the CGI program as text. The format in which the field and its value are sent to the CGI program is as follows:

&AMOUNT=24.95

If AMOUNT is defined in the database format file as a Packed(7,2) field, QtmhCvtDb automatically converts the character text "24.95" to X'002495F' and copies it into the data field in the data structure.

For more information on the format of data sent to CGI programs, see *URL-encoded strings*.

Table 13.7: QtmhCvtDb – API Parameters

Parameter	Definition	Description
Database file name	Char(20) – Input	The first 10 positions of this parameter contain the name of a database file used as a format file. The second 10 positions of this parameter should contain the library containing the file.
URL-encoded string	Char(*) – Input	The URL-encoded string retrieved from either the QUERY_STRING environment variable or a call to the QtmhRdStin API. This should be an unprocessed URL-encoded string sent to the CGI program by the Web browser.
Length of the URL-encoded string	Int(4) 10i0 - Input	A 4-byte integer that indicates length of the URL-encoded string. Retrieve this value by calling QtmhGetEnv and passing to it the CONTENT_LENGTH symbol.
Return value (e.g. data structure)	Char(*) – Output	A character field, normally an externally described data structure that receives the converted data. The API converts the URL-encoded string from parameter 2 into a usable format by using the database file specified on parameter 1. The result is stored in the field passed on this parameter.
Length of the return value's field.	Int(4) 10i0 – Input	Length of the return field. If the return value is a field or data structure, use the %SIZE() built-in function to retrieve the length of the field or data structure.
Length of return data "bytes returned"	Int(4) 10i0 – Output	Length of the data returned to the return value parameter.

Table 13.7: QtmhCvtDb – API Parameters

Parameter	Definition	Description
Response code	Int(4) 10i0 – Output	A response status code that indicates if a conversion problem was detected. The valid response codes are: 0. No response code generated. 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. Note: All response codes are returned as negative values.
Standard API structure	Char(*)	A data structure that is passed by reference. It should be a standard OS/400 API error data structure.

A prototype for QtmhCvtDb is shown in Example 13.8.

Example 13.8: Prototype for QtmhCvtDb

```
.....DName+++++++++++EUDS.......Length+TDc.Functions++++++++++++++++
     D QtmhCvtDb       PR                  ExtProc('QtmhCvtDb')
     D  fmtFile                      2ØA   Const
     D  urlString                65535A   Const OPTIONS(*VARSIZE)
     D  nURLLen                     1ØI Ø Const
     D  RtnDS                    65535A   Options(*VARSIZE)
     D  nRtnDSLen                   1ØI Ø Const
     D  nBytesRtn                   1ØI Ø
     D  nAPIRtnCode                 1ØI Ø
     D  QUSEC                              Like(QUSEC)
```

Example 13.9 illustrates how to call the QtmhCvtDb API.

Example 13.9: Calling the QtmhCvtDb API

```
.....DName+++++++++++EUDS.......Length+TDc.Functions+++++++++++++++++++
     D MyFile          DS                  QUALIFIED
     D  dbFile                       10A
     D  dbLib                        10A

     D urlData         S           4096A
     D OrderRec      E DS                 EXTNAME(ORDFILE)
     D nInStrLen       S             10I 0
     D nRtnDataLen     S             10I 0
     D nReplyCode      S             10I 0

.....Clrn01..............OpCode(ex)Extended-factor2++++++++++++++++++++++++
     C                   Eval      dbFile = 'ORDFILE'
     C                   Eval      dbLib  = 'CGILIB'
     **  Get the length of the CGI URL-encoded string
     C                   Eval      InStr = CGIData
     C                   Eval      nInStrLen = %Len(%TrimR(InStr))
     C                   Eval      nOutDSLen = %size(OutStr)
     C                   callp     QthmCvtDB(myFile : urlData :
     C                                   %len(%TrimR(urlData)):
     C                                   orderRec : %size(orderRec):
     C                                   nRtnDataLen : nReplyCode :
     C                                   apiError)
```

URL ENCODING

Data sent to a CGI program, either through the environment (when METHOD="GET") or through standard-input (when METHOD="POST") is received by the CGI program in URL-encoded format. URL-encoded format is simply a long character string containing the URL (Web address) of the Website's domain, followed by the CGI program name and the data typed into the HTML form.

The domain URL and CGI program name are separated from the form's data by a question mark (?). A typical URL-encoded string might look like the following:

http://www.rpgiv.com/cgi-bin/mycgi.pgm?ACCT=12345&NAME=Bob+Cozzi&LOC=Chicago

Note the question mark (?) between the program name and the HTML form data that follows it. Each field from the HTML form appears in the URL-encoded string immediately following the question mark. Each form field following the first field name is prefixed with an ampersand (&) character. The text assigned to the field is separated from the field name by an equals sign (=).

Any blanks (spaces) embedded in the URL-encoding string are replaced with plus signs (+). What happens if a plus sign is embedded in a field? It is converted into the hexadecimal equivalent of the plus sign. In fact, all non-alphanumeric characters are replaced with the hexadecimal equivalents. These hexadecimal values are referred to as *escape characters* or *escape sequences*, and are represented with a percent sign (%) followed by two hexadecimal characters.

For example, if the CITY field on an HTML form contains “Glen Ellyn, IL” and the AGE field contains “30", a URL-encoded string similar to the following is sent to the CGI program.

http://www.domain.com/dir?CITY=Glen+Ellyn%2E+IL&AGE=30

In the previous example, the comma is converted to hexadecimal x’2E’ and the blanks are converted to plus signs (+).

atoll() – C Runtime Library Function

The atoi() and atoll() C language runtime library functions convert a character string containing numeric data into an integer. The integer value can be assigned to a numeric field of any data type in RPG IV. The atoi() (ASCII to integer) function supports up to a 10-digit numeric value up to about 2.14 billion, whereas atoll() (ASCII to long long) supports up to a 19 digits.

When using C runtime functions, the binding directory QC2LE must be specified when compiling the source member. Specifically, BNDDIR(QC2LE) on the CRTRPGMOD or CRTBNDRPG commands is required. A more effective option is to add this compiler parameter to the source code itself. Add a header specification entry that includes the BNDDIR(‘QC2LE’) keyword Note that when a compiler parameter is used on the header specification, its parameters should be enclosed in single quotation marks and normally are in all uppercase. The prototype for these two functions in shown in Example 13.10.

Example 13.10: Prototypes for C runtime functions

```
.....DName+++++++++++EUDS.......Length+TDc.Functions++++++++++++++++++++
     H BNDDIR('QC2LE')

     D atoi            PR            10I 0 ExtProc('atoi')
     D  dInput                         *   VALUE Options(*STRING)

     D atoll           PR            20I 0 ExtProc('atoll')
     D  dInput                         *   VALUE Options(*STRING)
```