INFORMIX®-4GL CGI Library

Version: 4.16 UC1 and 6.04 UC1
August 1996
Part No. 000-8952

Published by: Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025

Contents

Introduction to the INFORMIX-4GL CGI Library

The INFORMIX-4GL CGI Library is a function library that extends the existing functionality of the INFORMIX-4GL product to enable you to build web-based applications (CGI scripts) that interact with databases.

You can use the functions in the 4GL CGI Library to dynamically produce web pages based on data held in your corporate databases. Your applications can interact with a database in any way you like, such as querying the database and displaying the results, or changing or updating the database.

You can dynamically format data found in your SQL databases for the web as HTML reports, forms, listings, catalogs, on-line documents and so on.

Who Should Read this Document

This document is intended for existing INFORMIX-4GL users who wish to learn how to build web-based INFORMIX-4GL applications.

This document does not teach how to use the standard INFORMIX-4GL functionality . Please see the printed INFORMIX-4GL documentation for information on INFORMIX-4GL.

This document also assumes that you understand the concepts of using the web, writing HTML tags, and using CGI scripts. However, for newcomers to the subject, we have included background information on HTML and CGI scripting, in the section Background To HTML and Gateway Scripts (page 39).

Overview of the INFORMIX-4GL CGI Library

The INFORMIX-4GL CGI Library contains functions that you can use inside a CGI script to process input from web-based forms, access CGI environment variables, and print data to an output file such as a web page.

Inside a CGI script, you have access to the full range of INFORMIX-4GL functionality.

Accessing Environment Variables and Form Values

The 4GL CGI Library includes several functions to read and interpret the information passed from an HTML form via CGI to your 4GL application.

- icgi_start() (page 25)

  This function initializes the application.
- icgi_getvalue(char *name) (page 26)

  This function gets the value of a form entry or an environment variable.
- icgi_getnvalue(char *name, int n) (page 26)

  This function gets the *n*th value of a form entry
- icgi_free() (page 27)

  This function frees the memory that was allocated to the form entry list.

Printing Data to the Output Document

The 4GL CGI Library includes three functions for printing to a Web page.

- icgi_mimetype(char *Mimetype) (page 27)

  This function sends the MIME type to the output Web page.
- icgi_print_text(char *Text) (page 28)

  This function prints text to the output document.
- icgi_print_blob(char* Mimetype, $loc_t Blob_loc) (page 28)

  This function prints Informix BLOB data to the output document.

Other CGI and 4GL Utility Functions

The 4GL CGI Library includes other useful functions for setting environment variables, URL-encoding and decoding strings, and cover functions for basic UNIX operations, such as cd and chmod.

Installing the INFORMIX-4GL CGI Library

Although anyone can download the INFORMIX-4GL CGI Library, only registered INFORMIX-4GL users will be able to install it and use it. The library is an add-on product to INFORMIX-4GL releases 4.16 UC1 and 6.04 UC1 for Unix, and only works when used in conjunction with the underlying INFORMIX-4GL software.

At present, the INFORMIX-4GL CGI Library is available only by downloading it from a web browser. It is not available through ftp.

Downloading and Installing the Library

1. To download the compressed tar file containing the 4GL CGI Library, select:

   `/cgi-bin/transmittal?2`
2. Copy the downloaded tar file to your INFORMIXDIR if it is not already there. (You may need to be logged in as root to do this step.)
3. Uncompress the tar file by using:

   ```
   uncompress 4glcgi.tarZ
   ```
4. Untar the tar file by using:

   ```
   tar -xf 4glcgi.tar
   ```
5. Run the installation process by using:

   ```
   ./installwk
   ```

The tar files include a file called `wkfiles` in the `etc` directory. This file contains the list of files that will be installed by the install script.

Using the INFORMIX-4GL CGI Library

This section discusses how to use the INFORMIX-4GL CGI Library.

The chapter uses an example script to illustrate the steps involved in writing a 4GL CGI script. The example discussed in this chapter presents a form that lets the user select a sport. The CGI script displays information about all the products in the Stock table whose description contains the name of the chosen sport.

The form has a text field called *YourName* and a set of radio buttons called *RB1*. The name-value pair for a set of radio buttons consists of the name of the set and the value of the selected button.

The example script uses a REPORT formatter to display the results of the query in a single table. The output page not only shows the results of the query, but also displays the form again, so the user can submit another query without going back a page in the web browser.

The script is invoked by a pre-script, which sets the values of the Informix-specific environment variables.

Execute the Script:

`/cgi-bin/pre-scriptfg`

View the Pre-script:

`pre-scriptfg.htm`

View the Script:

`scriptfg.4gl`

The Steps Needed for an ESQL/C CGI Script

The section A 4GL CGI Template (page 22) provides a template for a 4GL CGI script.

Setting Up

At the top of your script, define the globals needed by the script, if any. The example script, *scriptfg.4gl*, defines globals to represent the quote sign (’), the request method, the script name, the value of the text field containing the user’s name, the sport that the user selected, and the record for the database query:

```
GLOBALS
DEFINE
    g_quote             CHAR(1),
    g_request_method     CHAR(12),
    g_scriptname         CHAR(60),
```

```
    g_dear_user         CHAR(30),
    g_sport_chosen      CHAR(30)
DEFINE
    p_sport_prod RECORD
        description     CHAR(15),
        stock_num         INTEGER,
        manu_code         CHAR(3),
        qty_on_hand     INTEGER,
        unit             CHAR(4),
        unit_price         MONEY(6,2))
    END RECORD
END GLOBALS
```

Starting the CGI Processing

Call the icgi_start() function to initialize the structures containing the values of the environment variables and form field values:

```
MAIN
    IF (icgi_start() != 0 ) THEN
        CALL html_error_msg()
    END IF
```

Sending the MIME Type

Send the MIME type for the output document before writing anything to the output document:

```
CALL icgi_mimetype("text/html")
```

Sending the Title

Before writing data to the body of the Web page, you need to send the header information, such as the <HTML>, <HEAD> and <BODY> tags. Use the icgi_print_text() function to print the header information:

```
CALL icgi_print_text("<HTML><HEAD>")
CALL icgi_print_text("<TITLE>Welcome to the Sports
Database</TITLE>")
CALL icgi_print_text("</HEAD><BODY>")
```

Getting the Values of Environment Variables

Use the icgi_get_value() function to get the values of environment variables. For example, if you want the script to perform different tasks depending on the request method, you need to get the value of the REQUEST_METHOD environment variable. If your script presents a form that re-invokes the script, you need to get the value of the SCRIPT_NAME environment variable:

```
LET g_quote = "\""
LET g_request_method = icgi_getvalue("REQUEST_METHOD")
LET g_script_name = icgi_getvalue("SCRIPT_NAME")
LET g_script_name = g_quote, g_script_name CLIPPED, g_quote
```

Getting the Values of Form Elements

Use the icgi_getvalue() function to get the value of a form element. The following code shows how to get the values of the *YourName* field and *RB1* fields. The *YourName* field is a text field, and *RB1* is a set of radio buttons, where each radio button specifies a kind of sport, such as football or tennis:

```
LET g_dear_user = icgi_getvalue("YourName")
LET g_sport_chosen =  icgi_getvalue("RB1")
```

You can use the function icgi_getnvalue() to get the *n*th value of a form element. This would be applicable if the form has multiple elements with the same name, or the element is a multiple selection list, which can return multiple values.

The function icgi_getvalue() returns an empty string if the specified form entry does not exist, or has no value. Similarly, icgi_getnvalue() returns an empty string if the form entry does not exist or does not have an *n*th value. For example:

```
/* If no sport is selected, print a warning msg*/
IF (LENGTH (g_sport_chosen) == 0) THEN
    CALL icgi_print_text("<H2>Incomplete Form</H2> ")
```

Deciding What Task To Perform Based On How the Script Was Invoked

You can write your script so that its behavior depends on whether the script was invoked directly by opening its URL, or was invoked when a form was submitted.

You can use the REQUEST_METHOD to determine how the script was invoked. If the request method is "GET", the script was invoked directly from the URL rather than through the form. If the request method is "POST", the script was invoked from the form. (This last statement is true only if the form uses the POST method.)

The following code shows how the script determines what action to take, based upon the REQUEST_METHOD.

```
CASE g_request_method
    WHEN "GET"
      # called without a form, so simply output the query
form
      CALL sport_product_form_long()
    WHEN "POST"
      # the post method means a form has been submitted,
      # so process the results
      IF (LENGTH (g_sport_chosen) == 0) THEN
         CALL icgi_print_text("<H2>Incomplete Form</H2> ")
         CALL icgi_print_text("<P>Please select a sport
before ")
```

```
         CALL icgi_print_text("you press the Submit
button.</P>")
      ELSE
         # if a sport was selected, submit the query to the DB
         CALL query_sport()
      END IF
      CALL icgi_print_text("<BR><HR><BR><H2>")
      CALL icgi_print_text("Would you Like to Submit Another
Enquiry?</H2>")
      # show the form without the intro
      CALL sport_product_form_short(g_dear_user)
    OTHERWISE
      CALL html_error_msg()
END CASE
```

Interacting With Databases

You can use the standard 4GL functions to interact with databases. In the example script, the query_sport() function uses the value sent by the form to submit a query to the database, and then displays the results as a table formatted by a REPORT formatter:

```
WHENEVER ERROR CONTINUE
LET p_where_clause = "description MATCHES \"",
                        "*", g_sport_chosen CLIPPED, "*",
                        "\"",
                        " ORDER BY stock_num"
LET p_where_clause = p_where_clause CLIPPED, ";"
LET p_sql_stmt = "SELECT description, stock_num, manu_code,
",
                "unit, unit_descr, unit_price",
                " from stock WHERE ",
                p_where_clause
# use the stores7 database
```

```
DATABASE stores7
PREPARE sports_prod FROM p_sql_stmt
```

The function then creates a cursor, and uses it to iterate over the results of the query and send each record in the query to the REPORT formatter, which prints a table.

```
DECLARE prod_curs CURSOR FOR sports_prod
# output sports product list to report
LET p_row_cnt = 0
START REPORT rpt_sport_prod_list
FOREACH prod_curs INTO g_sport.*
    OUTPUT TO REPORT rpt_sport_prod_list(g_sport.*)
    LET p_row_cnt = p_row_cnt + 1
END FOREACH
FINISH REPORT rpt_sport_prod_list
```

If no records were returned by the query, html_no_rows() is called to print an explanation.

```
IF ( p_row_cnt = 0 ) THEN
    CALL html_no_rows()
END IF
```

Printing Data to the Output Page

To print data to a Web page, use the icgi_print_text() function. The example script uses this function extensively to display its results. For example, the following code shows the text that introduces the results of the query:

```
CALL icgi_print_text("<H1>Search Results</H1>")
CALL icgi_print_text("<P><B>Hello ")
CALL icgi_print_text(g_dear_user)
CALL icgi_print_text("</B>. This is what the search revealed.
")
CALL icgi_print_text("We have the following items in our
store:</P>")
```

This rest of this section discusses how to use REPORT formatters and print forms in detail. If you would like to skip these details, move on to Cleaning Up.

Using REPORT Formatters

To use a REPORT formatter to format results in a 4GL CGI script, call icgi_print-text() from inside the report.

The code for the REPORT formatter in the example script is shown below. It displays the results in an HTML table. The PAGE HEADER prints the HTML table heading. An HTML table row is printed ON EVERY ROW of the report. At the end of each page, the PAGE TRAILER prints a </TABLE> tag to close the HTML table.

```
REPORT rpt_sport_prod_list(p_sport_prod)
    DEFINE
        p_sport_prod       RECORD
        description        CHAR(15),
        stock_num          INTEGER,
        manu_code          CHAR(3),
        unit               CHAR(4),
        unit_descr         CHAR(15),
        unit_price         MONEY(6,2)
        END RECORD
    DEFINE
        p_string CHAR(2048)
    OUTPUT
        TOP MARGIN 0
        BOTTOM MARGIN 0
        PAGE LENGTH 200
    FORMAT
        FIRST PAGE HEADER
            CALL printColumnHeadings()
        PAGE HEADER
```

```
            CALL icgi_print_text("<BR>")
            # printColumnHeadings starts the table
            CALL printColumnHeadings()
        ON EVERY ROW
            # start the HTML table row
            CALL icgi_print_text("<TR>")
            # print the table cell data
            # description
            CALL icgi_print_text("<TD>")
            CALL icgi_print_text(p_sport_prod.description)
            CALL icgi_print_text("</TD>")
            # stock_num
            LET p_string = p_sport_prod.stock_num USING
"<<<<<<<<<"
            CALL icgi_print_text(p_string)
            CALL icgi_print_text("</TD>")

            # manu_code
            CALL icgi_print_text("<TD>")
            CALL icgi_print_text(p_sport_prod.manu_code)
            CALL icgi_print_text("</TD>")

            # unit
            CALL icgi_print_text("<TD>")
            CALL icgi_print_text(p_sport_prod.unit)
            CALL icgi_print_text("</TD>")

            # unit_descr
            CALL icgi_print_text("<TD>")
            CALL icgi_print_text(p_sport_prod.unit_descr)
            CALL icgi_print_text("</TD>")
```

```
            # unit_price
            CALL icgi_print_text("<TD>")
            CALL icgi_print_text(p_sport_prod.unit_price)
            CALL icgi_print_text("</TD>")

            # end the table row
            CALL icgi_print_text("</TR>")

    PAGE TRAILER
            # end the table
            CALL icgi_print_text("</TABLE>")
END REPORT
```

Printing Forms

You can print forms just as you print any other kind of information, by using the icgi_print_text() function. This section describes how the example script prints a form displaying a text field, a set of radio buttons, a reset button, and a submit button. You can, of course, build a form that has many other kinds of elements. (Your script may not even display a form.) If you want to skip the detailed discussion of the functions in the example script that print the form, you can go directly to Cleaning Up (page 20).

The example script defines a function sport_product_form_long(), which prints the introductory text that is needed when the user opens this page for the first time. Then it calls the function sport_product_form_short() which prints the actual form, consisting of the text field, the radio buttons and the Submit button.

When the page is generated as a result of the user submitting the form, then the introductory text is not needed (since we presume that the user is now familiar with the form.) In this case, the main() function calls the sport_product_form_short() function directly.

Here is the code for these sport_product_form_long():

```
FUNCTION sport_product_form_long()
    DEFINE p_string CHAR(2048)
    # Form Title and Greeting
    CALL icgi_print_text("<H1>Welcome to the Sports
Database</H1>")
    CALL icgi_print_text("<P>This form lets you find the
descriptions of various ")
    CALL icgi_print_text("products in the Stock table of the
Sports Database. ")
    CALL icgi_print_text("You can find the description of all
the products related ")
    CALL icgi_print_text("to volleyball, basketball, tennis,
baseball or football. ")
    CALL icgi_print_text("That is, you can find the products
whose description ")
    CALL icgi_print_text("includes the name of these
sports.</P>")
    CALL icgi_print_text("<BR>")
    CALL sport_product_form_short("Your_name")
    RETURN
END FUNCTION
```

Printing the Form Itself

The sport_product_form_short() function prints the form.

The action for the form is the script itself, so that when the user submits the form, the script is invoked again.

```
FUNCTION sport_product_form_short()
    DEFINE p_string CHAR(2048)
    CALL icgi_print_text("<FORM METHOD=\"POST\" ")
    CALL icgi_print_text("ACTION=")
    CALL icgi_print_text(g_script_name)
    CALL icgi_print_text(" >")
```

Displaying the Text Field

The first element in the form is a text field called *YourName*, where the user can enter their name if they want.

```
CALL icgi_print_text("<P>Please enter your name</P>")
CALL icgi_print_text("<INPUT TYPE=\"text\" NAME=\"YourName\"
SIZE=30 VALUE=")
CALL icgi_print_text(dearUser)
CALL icgi_print_text("</P>")
```

Displaying the Radio Buttons

The second element is a set of radio buttons, called *RB1*. Each radio button specifies a kind of sport, such as volleyball, basketball, tennis, baseball, and football.

```
CALL icgi_print_text("<P>What kind of sport are you
interested in?")
CALL icgi_print_text(" We will tell you about all the
products related")
CALL icgi_print_text(" to that sport that we carry in our
store.</P>")
```

The radio buttons are displayed in an unordered list.

```
CALL icgi_print_text("<UL>")
CALL icgi_print_text("<l1><input type=\"radio\" name=\"RB1\"
VALUE=\"volleyball\">volleyball<BR>")
CALL icgi_print_text("<l1><input type=\"radio\" name=\"RB1\"
VALUE=\"basketball\">basketball<BR>")
CALL icgi_print_text("<l1><input type=\"radio\" name=\"RB1\"
VALUE=\"tennis\">tennis<BR>")
CALL icgi_print_text("<l1><input type=\"radio\" name=\"RB1\"
VALUE=\"baseball\">baseball<BR>")
CALL icgi_print_text("<l1><input type=\"radio\" name=\"RB1\"
VALUE=\"football\">football<BR>")
CALL icgi_print_text("</UL>")
```

Displaying the Reset and Submit Buttons, and Ending the Form

The form has a Reset and a Submit button:

```
CALL icgi_print_text("<INPUT TYPE=\"submit\" VALUE=\"Submit
Query\">")
CALL icgi_print_text("</FORM> <HR>")
RETURN
END FUNCTION
```

This is what the form should look like:

Begin linear form:

Welcome to the Stores Database

This form lets you find the descriptions of various products in the Stock table of the Stores Database. You can find the description of all the products related to volleyball, basketball, tennis, baseball or football. That is, you can find the products whose description includes the name of these sports.

Please enter your name:

What kind of sport are you interested in? We will tell you about all the products relevant to that sport that we carry in our store.

- ❍ volleyball
- ❍ basketball
- ❍ tennis
- ❍ baseball
- ❍ football

Reset

Submit Now

End linear form.

Cleaning Up

After your script has finished sending data to the output file, it should close the Web page properly, by printing </BODY> and </HTML> tags, along with any other footer information you want.

Also, it is good policy to free the memory allocated to the environment variables and form field values by calling the icgi_free() function.

The following code illustrates the final lines of an example main() function:

```
CALL icgi_print_text("</BODY></HTML>")
LABEL main_exit:
CALL icgi_free()
END MAIN
```

Setting Informix-specific Environment Variables

The Informix-specific environment variables, INFORMIXDIR and INFORMIXSERVER, are not set automatically when a CGI script is invoked. You must take steps to set them.

The INFORMIXDIR variable must be set before running a compiled 4GL executable, such as a compiled CGI script.

If the INFORMIXDIR variable is not set at runtime, the executable uses the directory "/usr/informix" for INFORMIXDIR. If you have already installed the Informix products in another directory, you can create a symbolic link to /usr/informix, for example:

```
ln -s /usr/infmx7.10 /usr/informix
```

The INFORMIXSERVER environment variable must be set before you try to connect to a database.

You can use the function i4gl_setenv() to change environment variables from within the CGI executable. For example:

```
DEFINE env_str CHAR(256),
    return_value INTEGER
LET env_str = "INFORMIXSERVER=minnie"
```

```
LET return_value = i4gl_setenv(env_str)
```

The function i4gl_setenv() returns a non-zero value if the function is unable to alter the environment.

However, if you set the value of the INFORMIXSERVER environment variable within the body of the script, your compiled program will only work when placed on a server that can access the appropriate server.

Another option is to ensure that the administrator for the Web Server workstation where your script resides sets the Informix environment correctly for the server. However, some HTTP servers reset the environment at start-up. (We found the NCSA server resets the environment while the CERN server does not.)

You can also define a "pre-script" or wrapper script for your script. The pre-script can set the INFORMIXDIR and INFORMIXSERVER environment variables, and then call the script. Then if you move your program to another server, you simply need to edit the pre-script to point to the new INFORMIXDIR and INFORMIXSERVER, instead of having to edit your script and recompile it.

The following script is an example of a pre-script.

```
#!/bin/sh
DISPLAY = :0.0
INFORMIXDIR=/release/dir/7.20.LV3PJ
INFORMIXSERVER=minnie
export DISPLAY INFORMIXDIR INFORMIXSERVER
# Change this to reflect the script to be invoked
/release/dir/home/cgi-bin/scriptfg.cgi
#_____________________DONE_____________________________
```

To make an HTML form invoke the pre-script, specify the name of the pre-script as the action of the form, for example:

```
icgi_print_text("<FORM METHOD=\"POST\"
ACTION=\"/release/dir/home/cgi-bin/pre-scriptfg\">");
```

which generates the following HTML text:

```
<FORM METHOD="POST" ACTION="/release/dir/home/cgi-bin/pre-
scriptfg">
```

Note that the value of the SCRIPT_NAME environment variable is the script that CGI invoked, therefore if a script is invoked by a pre-script, the value of the SCRIPT_NAME environment variable is the name of the pre-script.

Compiling the Script

Compile your 4GL CGI script in the same way that you compile other 4GL scripts. For example:

```
c4gl -I${INFORMIXDIR}/incl sample1.4gl -o sample1.exe
-L${INFORMIXDIR}/lib -l4glcgi
```

HP note: If you are compiling on a Hewlett Packard system and are using the shared library, you also need to pass the `-Wl,+s` options to the compiler. These options , to specify that during runtime, the executable should look at the SHLIB_PATH environment variable first to find the shared library. For example:

```
c4gl -Wl,+s -I${INFORMIXDIR}/incl sample1.4gl -o sample1.exe
-L${INFORMIXDIR}/lib -l4glcgi
```

When the program is compiled, make sure it has the correct access privileges for anonymous users to execute. Then move it to the directory on your Web server where executable CGI programs reside. (This directory is usually called cgi-bin.) If you do not know where this directory is, ask your Web Server Administrator.

A 4GL CGI Template

The following code provides a basic template for a 4GL CGI application:

```
MAIN
    DEFINE myfield CHAR(50)
    # send the MIME-type first
    CALL icgi_mimetype("text/html")
    IF (icgi_start() == 0 ) THEN
         # send error message in HTML
         RETURN
```

```
    END IF
    # get the form entries
    LET myfield = icgi_getvalue("myfield")
    # construct SQL
    # execute SQL
    # print HTML #
    CALL icgi_print_text("<TITLE>My HTML Page</TITLE>")
    CALL icgi_print_text("<P>etc ... </P>")
    # clean-up and return #
    CALL icgi_free()
    RETURN
END MAIN
```

INFORMIX-4GL CGI Library Reference

Functions for Getting Environment Variable and Form Values

icgi_start()

This function initializes the CGI processing. Your script must call icgi_start() before retrieving environment variable or form values.

The function icgi_start() does the three tasks required to receive and interpret the data passed from CGI:

- Reads and stores the information from CGI specific environment variables.
- Reads the URL-encoded query string and decodes it.
- Parses the query string into a list of form entries.

This function return non-zero upon successful completion and zero upon failure:

```
IF ( icgi_start() == 0 ) THEN
# error handling code here
END IF
```

icgi_getvalue(char *name)

The function icgi_getvalue() returns the value of either an environment variable or a form element as a string. For example:

```
DEFINE request_method CHAR(4)
LET request_method = icgi_getvalue("REQUEST_METHOD")
DEFINE customer_num_field CHAR(15)
LET customer_num_field = icgi_getvalue("customer_num_field")
```

The icgi_getvalue() returns a NULL string if the field or variable is non-existent or is existent but does not have a corresponding value:

```
DEFINE fld1 CHAR(40)
LET fld1 = icgi_getvalue("field1")
IF (fld1 IS NULL ) THEN
     # handle empty field
END IF
```

Parameters:

name	The name of the form element or environment variable whose value you want to get.

icgi_getnvalue(char *name, int n)

The function icgi_getnvalue() returns the *n*th value of a form element as a string. Use this function when a form has multiple elements with the same name, or has a multiple selection list, which can return multiple values.

For example:

```
DEFINE val3 CHAR(15)
LET val3 = icgi_ngetvalue("colorchoice" 3)
```

If a form has multiple fields with the same name, the one closest to the top of the form is number 1; the second highest is 2, and so on.

The function icgi_getnvalue() returns a NULL string if the field is non-existent or does not have an nth value.

Parameters:

name	The name of the form element or environment variable whose value you want to get.
n	The number of the field whose value is to be retrieved, for example, 2, for the second occurrence.

icgi_free()

The function icgi_free() frees the memory that was allocated to storing variables and form entries:

```
MAIN
IF ( cgi_start() == 0 ) THEN
    # handle error
END IF
# continue CGI application
LABEL main_exit:
     CALL cgi_free()
END MAIN
```

Functions for Printing Data to a Web Page

icgi_mimetype(char *Mimetype)

This function prints the MIME-type (also known as "Content Type") for the output document.

When sending a document to be displayed in a web browser, the MIME-type should be the sent first, so the Web browser knows what kind of content to expect.

Call the function icgi_mimetype() with a valid MIME-type before attempting to print anything else to the output document. Typical HTML output would require the MIME-type "text/html", for example:

```
FORMAT
FIRST PAGE HEADER
    CALL icgi_mimetype("text/html")
```

Parameters:

Mimetype	A valid MIME type as a string.

icgi_print_text(char *Text)

The function icgi_print_text() prints text to the output document. The function appends a new-line to the end of the text.

The following 4GL statements...

```
CALL icgi_print_text("<B>")
CALL icgi_print_text("Bolded Text")
CALL icgi_print_text("</B>")
```

... results in the following HTML source text:

```
<B>
Bolded Text
</B>
```

which appears in the browser as:

Bolded Text

Parameters:

Text	The text to be printed.

icgi_print_blob(char* Mimetype, $loc_t Blob_loc)

The function icgi_print_blob() prints Informix BLOB data to the output document (BLOB data types are available with the OnLine engine). The function takes a string containing a valid MIME-type and a BLOB variable (TEXT or BYTE). The function will handle either type of BLOB, whether it is located in memory or a file.

```
DEFINE mime_type CHAR(30),
         image_blob BYTE
    LOCATE image_blob IN MEMORY  # can also be located in file
    SELECT image_blob_col
        FROM image_table
        INTO image_blob
        WHERE primary_key = 1000
    LET mime_type = "image/gif"
    CALL icgi_print_blob(mime_type, image_blob)
    FREE image_blob
```

You must locate the BLOB correctly prior to fetching it from the database. For more information about accessing BLOB data from 4GL, refer to the INFORMIX-4GL Reference Manual.

If the output is a BYTE BLOB, this function is typically called "stand-alone" without prior or subsequent HTML text, therefore a valid MIME-type should be used. For example, if the data is a GIF image:

```
CALL icgi_print_blob("image/gif", my_gif_blob)
```

If the output is a TEXT BLOB, this function might be called amongst other HTML text (using calls to icgi_print_text()), therefore the MIME-type has already been sent using icgi_mimetype(). In this case, pass a NULL or empty string. For example:

```
LET mime_type = NULL
CALL icgi_print_blob(mime_type, my_text_blob)
```

Parameters:

Mimetype	A valid MIME type as a string
Blob_loc	A location for a Blob

Other CGI Utility Functions

icgi_decode(char *string)

This function decodes a URL-encoded string and returns the decoded string.

```
DEFINE decoded_str CHAR(2048),
    encoded_str CHAR(2048)
    LET decoded_str = icgi_decode(encoded_str)
```

Parameters:

string	The string to decode.

icgi_encode(char *string)

The function icgi_encode() encodes a plain text string using URL-encoding rules and returns the encoded string.

```
DEFINE decoded_str CHAR(2048),
encoded_str CHAR(2048)
LET encoded_str = icgi_encode(decoded_str)
```

Parameters:

string	A string to encode using URL-encoding rules.

Other 4GL Utility Functions

i4gl_setenv(char *string)

The function i4gl_setenv() allows you to change environment variables from within the CGI executable. If the function is unable to alter the environment, it returns a non-zero value.

```
DEFINE env_str CHAR(256),
    return_value INTEGER
    LET env_str = "INFORMIXSERVER=garfield"
    LET return_value = i4gl_setenv(env_str)
```

Parameters:

string	A string consisting of the environment variable and the value to set it to.

i4gl_access(char *file, char *permissions)

The function i4gl_access() allows the 4GL application to query the accessibility of a file. The first parameter is a string containing the file name. The second parameter is a string containing one, some or all of the following characters:

- "r" to check whether the file is readable
- "w" to check whether the file is writable
- "x" to check whether the file is executable

The function returns an integer. The returned value is the same as for the UNIX/C access() function.

```
DEFINE return_value INTEGER,
        access CHAR(3),
        file_name CHAR(250)
    LET access = "w"
    LET file_name = "myfile.htm"
    LET return_value = i4gl_access(file_name, access)
```

Parameters:

file	The name of a file.
permissions	A string consisting of any combination of "r", "w", and "x", indicating the permissions to be checked.

i4gl_cd(char *directory)

The function i4gl_cd() allows the 4GL application to change the current working directory. It returns zero if successful, non-zero if unsuccessful.

```
DEFINE return_value INTEGER,
    directory CHAR(256)
    LET directory = "/www/html"
    LET return_value = i4gl_cd(directory)
```

Parameters:

directory	The directory to switch to as the current working directory.

i4gl_chmod(char *filename, char* mode)

The function i4gl_chmod() changes the permissions of a file. The mode is the same numeric permissions (read / write / execute by owner / group / other) that can be passed to the Unix utility chmod. It returns the same code number that the chmod utility returns.

```
DEFINE return_value INTEGER,
    file_name CHAR(60),
    mode CHAR(4)
    LET file_name = "myfile.htm"
    LET mode = "664"
    LET return_value = i4gl_chmod(file_name, mode)
```

Parameters:

filename	A string of the filename whose mode is to be changed.
mode	A string indicating a permission mode.

i4gl_getpid()

The function i4gl_getpid() returns an integer indicating the current process ID.

```
DEFINE pid INTEGER,
    LET pid = i4gl_getpid()
```

i4gl_pwd()

The function i4gl_pwd() returns a string indicating the current working directory.

```
DEFINE directory CHAR(256)
    LET directory = i4gl_pwd()
```

i4gl_rm(char *filename)

The function i4gl_rm() deletes a file from the file system. It returns the same code number that the Unix unlink utility does.

```
DEFINE filename CHAR(60)
    DEFINE return_value INTEGER
    LET filename = "/mydir/tmpfile.htm"
    LET return_value = i4gl_rm(filename)
```

Parameters:

Filename	A string of the name of the file to be deleted.

i4gl_tmpfile(char *bufferForTmpFile)

The function i4gl_tmpfile() generates a filename with a fully qualified path that can be used as a temporary file. The argument *bufferForTmpFile* can be NULL. This function provides the same functionality as the UNIX/C tmpname facility.

```
DEFINE path CHAR(256)
    LET path = i4gl_tmpfile()
```

Parameters:

bufferForTmp-File	The name of a file to use as a temporary file. This can be NULL, in which case a default temp file is generated.

Debugging and Trouble-Shooting

This chapter includes the following:

Debugging Tips

The following steps are recommended for testing your program:

Make sure the program runs from the command line.

When the program is run from the command line, you should see the text generated for the output document, including the HTML tags. If your program processes input received from a form, it will not do much if it is invoked from the command line, but at least you should see the text for the start of the output document.

The icgi_start() function checks if the REQUEST_METHOD and SERVER_NAME environment variables are set. If they are, it returns TRUE, otherwise it returns FALSE. If the progress of your program depends on the icgi_start() function being successful, you must set the REQUEST_METHOD and SERVER_NAME environment variables before invoking your program in the command line.

If your program processes values sent by a form, you can hardwire the values in the program, or pass them from the command line.

Run the program from the command line as an anonymous user .

This simulates how the Web server invokes a CGI program. You personally may have set up your environment to define INFORMIXDIR and INFORMIXSERVER, or to define other parameters that the CGI script uses. However, when the Web server invokes a CGI script, it will use the same environment settings that an anonymous user would have.

Use printing functions to track the progress of the program.

You can use the function icgi_print_text() to print data as the program runs.

If you run the program from the command line, the printing functions send their output to the command line window. If you invoke the program from a Web browser, the printing functions send their output to the generated Web page.

Invoke the program from a Web browser.

If the program runs successfully from the command line, copy it to your CGI executable directory, and try invoking it from a Web browser. If it does not work, check the following:

- Does the program have permissions set for Web users to execute it?
- Are the INFORMIXDIR and INFORMIXSERVER environment variables set correctly? You can use a pre-script to set these environment variables and then invoke the program. For example:

```
#!/bin/sh
DISPLAY = :0.0
INFORMIXDIR=/release/dir/7.20.LV3PJ
```

```
INFORMIXSERVER=minnie
export DISPLAY INFORMIXDIR INFORMIXSERVER
# Change this to reflect the script to be invoked
/release/dir/home/cgi-bin/script11.cgi
#_____________________DONE_____________________
____
```

If you use a pre-script, the value of the SCRIPT_NAME environment variable is the pre-script.

General Hints

You cannot print befor the header or after the footer have been processed

If the expected text does not appear in the output document, check that your script did not try to print data before sending the MIME type and printing header information (such as <HTML><BODY>) or after printing footer information (such as </BODY></HTML>).

You can use SCRIPT_NAME to refer to the current script

If your script displays a form that re-invokes the same script, you can use the value of the SCRIPT_NAME environment variable as the value of the form. To start a form, use the ixHtmlDoc::beginForm() function, specifying the script to use as the action of the form. For example:

```
icgi_print_text("<FORM METHOD=\"POST\" ");
icgi_print_text("ACTION=");
icgi_print_text(icgi_getvalue("SCRIPT_NAME"));
icgi_print_text(" >");
```

If you use a pre-script, the value of the SCRIPT_NAME environment variable is the pre-script.

Troubleshooting

When you tryto invoke the script from a Web browser, you get a "Document contains no data" error message.

> Are you invoking a pre-script that sets INFORMIXDIR and INFORMIXSERVER before invoking the program? Is your pre-script executable? Is your program executable? Does your pre-script call the correct script?

When you try to invoke your program, the Web browser never returns.

> This often indicates an uncaught error. Stop the process (by going back a page in the browser if necessary). If possible, check the processes on the server and kill the dead process (so as not to burrden the server). Try running the program from the command line to find out what is wrong.

Your form is not active – You press the Submit button and nothing happens.

> Did you end the form?

Your script submits a query to a database based on values received from a form, but the query never finds any matches when you think it should.

> Database queries are case sensitive, for example "*Ball*" does not match "tennisball." Check that the case of the form value matches the case of the items in the database. Be sure to check the case of the value returned by the form element, not the label of the form element. For example, the label of the form element, such as a radio button, might be "tennis", whereas the return value is "Tennis."

Background To HTML and Gateway Scripts

This section briefly discusses HTML, web-based forms, and gateway scripts, and lists a variety of sources for more information.

Introduction to HTML

HTML is a markup language for creating pages to be viewed on the world wide web. To view pages on the web, you need a browser such as Netscape, but presumably since you are reading this page on the web, you have access to a browser.

If you have never seen a sample HTML file, select the **View** menu in your browser's menu bar, and choose the **Document Source** option. A file will open to show the HTML source text for this page. This source text is more complicated than HTML source text usually needs to be, since every paragraph has a name reference (so that the index can find paragraphs). However, it should give you some idea of what HTML source text looks like.

To specify the layout of text in a web browser, you use inline tags such as <P> to start a paragraph and </P> to end a paragraph. Other examples are:

<H1> starts a first level heading

</H1> ends a first level heading

<H2> starts a second level heading

</H2> ends a second level heading

Creating Forms on the Web

The purpose of the Informix Webkits is to enable you to build Web-based applications that interact with databases. Often you will want your application to interact with the user, for example, to let the user specify values to use in a database query. This section discusses how to use forms to solicit input from the users.

This section contains the following subsections:

Using the <FORM> Tag

A form starts with the <FORM> tag, and ends with the </FORM> tag. All interface elements in the form must be located between the start of the form and the end of the form. If you want a form to do something useful with the information selected by the user, the form must contain a **Submit** button.

The <FORM> tag specifies the request method and the action to be invoked by the Submit button. For example:

```
<FORM METHOD = "POST" ACTION = "/cgi-bin/landscp1"
```

Interface Elements in a Form

A form contains interface elements that allow the user to make choices and enter information. These interface elements include:

Begin linear form:

Text Fields

A text field is a one-line area that the user can type into. You can specify the size of the field.

Please enter your name:

Text Areas

A text area is a multi-line area where a user can type into.

Please enter your address:

This is where you live.

Checkboxes

A checkbox acts as a toggle; if you press it when it is on, it switches off, and vice versa. Checkboxes can be nested in lists.

Choose the plants you would like in your yard:

- ❑ Shrubs
  - ❑ Azaleas
  - ❑ Rhodedendrons
- ❑ Flowers
  - ❑ Roses
  - ❑ Sunflowers
  - ❑ Marigolds
  - ❑ Nasturtiums

Radio buttons

Radio buttons can be grouped together to form a set of choices. Only one choice per set can be selected. When you select another radio button in the set, the current choice is selected, and the previous choice is deselected. Radiobuttons can be nested in lists.

Choose the kind of fence you would like:

- ❍ Painted Fence
  - ❍ Natural Stain
  - ❍ White Fence
  - ❍ Green Fence
- ❍ Wire Fence
- ❍ No Fence

Pop-up menus

A pop-up menu displays up all the elements in the menu when you click on it.

Please indicate the climate where you live:

- Wet winters, dry summers
- Rainy all year round
- Drought conditions most of the time
- Moderate winters, monsoonal summers

Fixed menus

For a fixed menu, you need to specify the size, which is the number of items that are displayed in a scrolling box.

What yard theme do you prefer:

- Mostly lawn with a few flower beds
- Natural shrubs, native wildflowers
- Manicured lawn and hedges
- Flowers everywhere
- The woodsy effect with lots of trees
- Easy care paving with flowers in pots
- Orchard look - lots of fruit trees
- Vegetable garden
- Herb garden
- Patio with gazebo

Reset buttons

When a Reset button is pressed, it sets all the form elements back to their initial values.

Reset

Submit buttons

If you want your form to do anything useful, it must have a **Submit** button.

When the user presses the **Submit** button, it invokes the action specified by the form. If a form does not have a **Submit** button, the user can select items in the form to their hearts content, but the form will never do anything.

The <FORM> tag has an option that allows you to specify what action is invoked when the user "submits" the form. The action can open another web page without using the information selected in the form, which is usually not very useful. The more common specification for the action is to open a URL that points to a "gateway script."

Submit Now

End linear form.

Forms can have other elements that are not discussed here, such as hidden fields and password fields. Please see an HTML reference source for a full description of all form elements. The section Further Information on HTML (page 44) lists several HTML reference sources on the Web.

Gateway Scripts

Gateway scripts are usually known as CGI (for common gateway interface) scripts. A CGI script can be written in any language that is supported by the server on which it resides. For example, on Unix servers you can run shell scripts, Perl scripts, C scripts, and scripts written with Informix Webkits.

The gateway script must decode the input from the form, and output a properly formatted Web page. You can use the Informix Webkits to write scripts that decode information sent by forms, use the information to submit queries to a database, and send the results of the query back to the user as HTML source code that makes a Web page.

Further Information

This section tells you where to find more information.

Further Information on the World Wide Web

- WWW FAQ

  http://www.boutell.com/faq/

- WorldWide Web Consortium's Style Guide for Online Hypertext

  http://www.w3.org/hypertext/WWW/Provider/Style/Overview.html

- NCSA Httpd Overview

  http://hoohoo.ncsa.uiuc.edu/docs/Overview.html

Further Information on HTML

- HTML Primer

  http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html

- Information about HTML

  http://union.ncsa.uiuc.edu/HyperNews/get/www/html.html>

- Composing Good HTML

  http://www.cs.cmu.edu/~tilt/cgh/

More Information about Gateway Scripts

- Information about the Common GateWay interface (CGI)

  http://hoohoo.ncsa.uiuc.edu/cgi/intro.html

- The Common Gateway Interface (CGI) Frequently Asked Questions (FAQ) List

  http://www.best.com/~hedlund/cgi-faq/faq.html

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

M

O

P

R

S

T

U

W