Net.Data Sample Macro Code

Here you will find answers to your "how-to" questions by providing examples that show techniques for untangling some of the "mysteries" of writing macros. The following examples assume that: (1) latest Net.Data code is loaded on your system; (2) Net.Data has been configured on your system; and (3) there are HTTP configuration directives to map "/cgi-bin/db2www/*" to the DB2WWW program in your CGI library.

General Language Features

Producing static HTML

Producing static HTML using Net.Data is really not very interesting, but it will illustrate to you that it is the **%HTML block** that determines what gets processed within a Web macro, and what data, if any, is returned to the client (e.g. browser).

Example 1

We want to first produce a Web page with the the words "Hello World". The Web macro below will do what we need:

```
------------------------------------------------------------
%HTML(HelloWorld1) {

  Hello World From HelloWorld1

%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC1, member HELLO1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac1.file/hello1.mbr/HelloWorld1
```

Example 2

You can combine the above two macros into one macro, and depending on the URL that is specified, you will see different output on your browser:

```
------------------------------------------------------------
%HTML(HelloWorld1) {
  Hello World From HelloWorld1
%}

%HTML(HelloWorld2) {

Hello World From HelloWorld2

%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC1, member HELLO2, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
         netdata.lib/basemac1.file/hello2.mbr/HelloWorld2
```

We can see the same output as we saw in example 2 by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
         netdata.lib/basemac1.file/hello3.mbr/HelloWorld2
```

Defining, referencing, and setting variables

There are three types of variables:

1. Explicitly defined variables using the **%DEFINE** statement.
2. Pre-defined variables, which are variables that are made available by Net.Data and are set to a value.
3. Implicitly defined variables, which are of four types:
    - Variables that are not explicitly defined but are instantiated when first referenced.
    - Parameter variables that are part of a **%FUNCTION block** definition and that can only be referenced within a %FUNCTION block.
    - Variables that get instantiated by Net.Data and correspond to form data or URL data name-value pairs.
    - Variables associated with a Net.Data table and that can only be referenced within a **%ROW block** or **%REPORT block**.

An **identifier** may be a variable or a function call. An identifier becomes **visible** (i.e. can be referenced) with its declaration or instantiation. The region where an identifier is visible is referred to as **scope**. There are five types of scope: **global**, **macro file**, **function block**, **report block**, and **row block**

An identifier has global scope if it can be referenced from anywhere within a macro file. Identifiers that are global in scope are: the built-in functions, form data, URL data, and variables instantiated from within an %HTML block.

An identifier has macro file scope if its declaration appears outside of any block. A block starts with an opening "{" and ends with "%}". (Note that %DEFINE blocks are excluded from this definition and should be treated as separate %DEFINE statements.) An identifier with macro file scope is visible from the point it is declared to the end of the macro file.

An identifier has function block scope if its declaration (i.e. parameters) or instantiation is located inside a %FUNCTION block (this also applies to variables that are referenced within a %REPORT or %ROW block but were not explicitly defined within the macro file or implicitly defined by Net.Data).

An identifier has report block scope if it can only be referenced from within a %REPORT block (e.g. table column names N1,...Nn). Only those variables that Net.Data implicitly defines as part of its table processing can have a report block scope. Any other variables that get instantiated will have function block scope.

An identifier has row block scope if it can only be referenced from within a %ROW block (e.g. table field values V1,...,Vn). Only those variables that Net.Data implicitly defines as part of its table processing can have a row block scope. Any other variables that get instantiated will have function block scope.

A reference to an identifier will result in that identifier being replaced with the value of the identifier. If a reference to a variable has no value associated with it, or a function call does not have a return value, the reference will get replaced by an empty string.

Example 1

The following macro illustrates variable scope. The macro includes a function block (scope_test) and an HTML block (define1). The function block expects two parameters, p1 (input-only) and p2 (output-only). It also contains a report block.

```
------------------------------------------------------------
%function(DTW_REXX) scope_test(IN p1, OUT p2)
{
  return
  %report {
    @dtw_assign(p2, "123")
    @dtw_assign(y, "yyy")


From within function block: x=$(x), y=$(y), p2=$(p2)
  %}
%}

%html(define1) {
@dtw_assign(x, "xxx")
@scope_test("xx", z)


From within html block: x=$(x), y=$(y), z=$(z)
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC2, member DEFINE1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac2.file/define1.mbr/define1
```

The first statement in the HTML block is a call to a built-in function that assigns the string "xxx" to an undefined variable x. Since x was not defined previously, Net.Data instantiates x with global scope, and assigns it the value of the string "xxx". The following statement is a function call to scope_test(), passing it two parameters, a literal string ("xx") and an undefined variable (z). Again, since z is undefined it gets instantiated with global scope and is assigned the value of the NULL string.

Within the function block, p1 has a value of "xx" and p2 has the value of the NULL string since it is an OUT parameter. In the report block, p2 gets assigned the string "123", and y gets assigned the string "yyy". Since p2 is an OUT parameter, upon the completion of the function

block the variable z will be set to the value of p2. The variable y is undefined and thus gets instantiated by Net.Data with function block scope since it is referenced within the function block.

Example 2

Here is a macro that defines two variables, VAR1 and VAR2, using the %DEFINE statement. It then references the variables in the %HTML block, including a variable that has not been previously defined (VAR3) and a predefined Net.Data variable (DTW_MACRO_FILENAME):

```
------------------------------------------------------------
%define VAR1 = "variable one value."
%define VAR2 = "variable two value."

%HTML(define2) {

The value of VAR1 is: $(VAR1)

The value of VAR2 is: $(VAR2)

The value of VAR3 is: $(VAR3)

The macro that is being processed is: $(DTW_MACRO_FILENAME)
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC2, member DEFINE2, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac2.file/define2.mbr/define2
```

Example 3

An alternate way to explicitly define variables is to define the variables in one %DEFINE block. Here is is the macro in example 1 written with one %DEFINE block:

```
------------------------------------------------------------
%define {
   VAR1 = "variable one value."
   VAR2 = "variable two value."
%}

%HTML(define3) {

The value of VAR1 is: $(VAR1)

The value of VAR2 is: $(VAR2)
```

```
The value of VAR3 is: $(VAR3)

The macro that is being processed is: $(DTW_MACRO_FILENAME)
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC2, member DEFINE3, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac2.file/define3.mbr/define3
```

Example 4

Below is an example of a macro accessing data sent to Net.Data via the URL route. You reference URL data like you would any other Net.Data variable:

```
------------------------------------------------------------
%HTML(define4) {

The value of P1 is: $(P1)

The value of P2 is: $(P2)
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC2, member DEFINE4, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
       netdata.lib/basemac2.file/define4.mbr/define4?P1=D1&P2=5
```

Example 5

You can do a lot of neat stuff with referencing variables from within a string value. For example, the following macro sets VAR1 to a string that references a URL data variable, and VAR2 to the lowercase equivalent of the URL data variable by using the **DTW_rLOWERCASE()** built-in function:

```
------------------------------------------------------------
%define VAR1 = "Form data variable is $(P1)"
%define VAR2 = "Form data variable in lowercase is @DTW_rLOWERCASE(P1)"

%HTML(define5) {

$(VAR1)

$(VAR2)
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC2, member DEFINE5, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
    netdata.lib/basemac2.file/define5.mbr/define5?P1=D1
```

Example 6

Another useful feature is defining a variable that results in a program being called by using the **%EXEC** form of the %DEFINE statement. In the macro below, anywhere the variable PGMVAR is referenced will result in the program "/QSYS.LIB/NETDATA.LIB/DOSOME.PGM" being called.

```
-------------------------------------------------------------
%define PGMVAR = %EXEC "/QSYS.LIB/NETDATA.LIB/DOSOME.PGM"

%HTML(define6) {

$(PGMVAR)
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC2, member DEFINE6, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
          netdata.lib/basemac2.file/define6.mbr/define6
```

Example 7

Changing the value of a Net.Data variable is done by using the built-in function **DTW_ASSIGN()**. The following Web macro switches the values of VAR1 and VAR2 so that VAR1 contains the value of VAR2 and VAR2 contains the value of VAR1:

```
-------------------------------------------------------------
%define VAR1 = "TRUE"
%define VAR2 = "FALSE"

%HTML(define7) {

BEFORE DTW_ASSIGN():  VAR1 = $(VAR1) and VAR2 = $(VAR2)

@DTW_ASSIGN(VAR1, VAR2)
@DTW_ASSIGN(VAR2, "TRUE")

AFTER DTW_ASSIGN(): VAR1 = $(VAR1) and VAR2 = $(VAR2)
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC2, member DEFINE7, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
         netdata.lib/basemac2.file/define7.mbr/define7
```

Simulating server-side includes

What we mean by server-side includes is the ability to specify a location in your macro where data will get inserted. The examples on this page will show you how to use the **%INCLUDE** statement, which allows you to specify a file that will be processed when encountered by Net.Data.

Example 1

Let us start with you wanting to put a common header and footer in all your Web pages. Typically, you would have to edit all your Web pages, adding HTML statements so that the header and footer that you desire get displayed when your Web page is loaded. And if the header or footer ever get changed (e.g. the company name is changed), then you would need to update all the Web pages again!

Assume the header and footer information is stored in the members HEADER and FOOTER, respectively, in library NETDATA and file BASEMAC3. The header and footer may contain any type of HTML data. The Web macro that includes the header and footer information may look like the following:

```
------------------------------------------------------------

%HTML(include1) {

%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC3.FILE/HEADER.MBR"

Welcome page

%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC3.FILE/FOOTER.MBR"

%}

------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC3, member INCLUDE1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac3.file/include1.mbr/include1
```

Thus, when you need to update the header and footer, you only need to update the HEADER and FOOTER files, changing the look of your site in a matter of minutes!

Example 2

Then there is the case where you have several files, and you want the files to be looked at as one file when a user clicks on a link. You have two options: (1) merge all the files into one, thus making the files bulky and unmanageable, or (2) use the %INCLUDE Net.Data construct to do what you want in the least disruptive manner as possible. Of course we choose option number two!

Say there are three chapters of a book that are in three separate members: CHAPTER1, CHAPTER2, and CHAPTER3. And these members are stored in file BASEMAC3, library NETDATA. Then the macro that "merges" the three chapters into one is:

```
------------------------------------------------------------

%HTML(include2) {

%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC3.FILE/CHAPTER1.MBR"

%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC3.FILE/CHAPTER2.MBR"

%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC3.FILE/CHAPTER3.MBR"


%}

------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC3, member INCLUDE2, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac3.file/include2.mbr/include2
```

Conditional processing

Conditional processing is the ability to conditionally evaluate a statement when the specified test expression evaluates to a non-zero value. In other words, the set of examples below show you how to use the **%IF**, **%ELIF**, **%ELSE**, **%ENDIF**, and **%WHILE** statements to return different HTML data depending on the value of a variable.

The important concept to note when using %IF and %WHILE statements is that the comparisons done in the test expression are treated as **integer** comparisons if the following conditions are true:

- the condition is a binary operation (<, >, <=, >=, !=, ==).
- both terms in the condition represent integers. This means the terms are strings of digits, optionally preceded by a '+' or '-' character. The string cannot contain any non-digit characters other than the '+' or '-'. Some valid strings are:

```
+1234567890
-47
000812
92000
```

The following strings are not valid:

```
-20       (contains blank characters)
234,000   (contains a comma)
57.987    (contains a decimal point)
```

If either of the terms in a condition does not represent an integer, then a normal string comparison will be done.

Example 1

The following Web macro will return different HTML text to the browser depending on the URL data that is sent:

```
------------------------------------------------------------
%HTML(cond1) {

%IF ( "$(COLOR)" == "G" )

The color value is GREEN.
%ELIF ( "$(COLOR)" == "R" )

The color is RED.
%ELSE

Color value is not known.
```

```
Please try again.
%ENDIF

%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC4, member COND1, we can reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac4.file/cond1.mbr/cond1?COLOR=G
```

Example 2

Here is a macro that determines what HTML page to display based on the HTTP_USER_AGENT environment variable, so one can generate data depending on the browser being used. We use the **%ENVVAR** construct to retrieve the value of the HTTP_USER_AGENT environment variable, and the **%INCLUDE** statement to reference the proper HTML page to display back to the browser.

```
------------------------------------------------------------
%define HTTP_USER_AGENT = %ENVVAR

%HTML(cond1) {

%IF (( "$(HTTP_USER_AGENT)" == "Mozilla/2.02 (OS/2; I)" ) ||
     ( "$(HTTP_USER_AGENT)" == "Mozilla/3.01 (X11; I; AIX 1))"

Show frame-enabled HTML.
%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC4.FILE/COND2A.MBR"
%ELSE

Show text-only HTML.
%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC4.FILE/COND2B.MBR"
%ENDIF

%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC4, member COND2, we can reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac4.file/cond2.mbr/cond2
```

Setting cookies

What is a **cookie**? It is basically data that is generated by a script running on the server and stored on a client's (e.g. browser) system. Cookies are usually used to store information about a client. The information may be use to keep track of a client's shopping selections; or a client's preferences, so that when next time the client returns the server script can generate a custom built HTML interface. Cookies are sent to the server by the browser based on the URL requested, and the script that gets invoked can access the cookies to determine what type of special action to take.

The current syntax of a cookie is as follows (words with all capital letters need to be replaced with values):

```
Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure
```

To access a cookie, the server script (Web macro) would need to reference the HTTP_COOKIE environment variable.

OK, enough about cookies. How do we do it in Net.Data? Read on.

Example 1

To set a cookie in a Web macro, you need to tell Net.Data not to print out the Content-type header. This is done by setting the DTW_PRINT_HEADER variable to "NO". Then, in the %HTML block that gets control, the first three lines must be the Content-type header, the Set-Cookie statement, and a blank line. The following example sets a cookie.

```
------------------------------------------------------------
%define DTW_PRINT_HEADER = "NO"

%html(cookie1) {
Content-type: text/html
Set-Cookie: UsrId=56, expires=Friday, 12-Dec-99, 12:00:00 GMT; path=/

Any text...
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC5, member COOKIE1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac5.file/cookie1.mbr/cookie1
```

Which would result in the cookie getting stored on the browser system if the browser supports cookies.

Example 2

When setting the cookie, you can include variable references and function calls in the cookie statement. The example below obtains the cookie name from a macro variable via a reference, and the cookie value is obtained from a macro variable reference that is passed to a Net.Data built-in function, **DTW_rURLESCSEQ**, so that blanks in the string are converted to %20.

```
------------------------------------------------------------
%define DTW_PRINT_HEADER = "NO"
%define cNam = "CookieName"
%define cVal = "NetData CookieValue"

%html(cookie2) {
Content-type: text/html
Set-Cookie: $(cNam)=@DTW_rURLESCSEQ(cVal),  path=/

Any html text
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC5, member COOKIE2, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac5.file/cookie2.mbr/cookie2
```

Example 3

You reference a cookie by referencing the HTTP_COOKIE environment variable. The following macro will display any cookie that was previously set:

```
------------------------------------------------------------
%define HTTP_COOKIE = %ENVVAR

%html(cookie3) {

The cookies that were set by the server are:
"$(HTTP_COOKIE)".
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file BASEMAC5, member COOKIE3, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac5.file/cookie3.mbr/cookie3
```

Setting INI-file path variables

On the IBM i, a Net.Data initialization (INI) file is not required. The benefit of using an INI file is shorter URLs and shorter references to programs and include files within your macro files. However, you are required to have an initialization file if you decide to create your own language environment.

They key to making file references (includes program references) are the MACRO_PATH, EXEC_PATH, and INCLUDE_PATH path statements.

Example 1

Assume in the INI file we added the statement:

```
MACRO_PATH    /QSYS.LIB/NETDATA.LIB/BASEMAC6.FILE
```

Then to reference the following Web macro (PATH1):

```
------------------------------------------------------------
%html(path1) {

This Web macro is $(DTW_MACRO_FILENAME).
%}
------------------------------------------------------------
```

we would load the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/path1.mbr/path1
```

If we did not have the MACRO_PATH initialization statement, then one would need to specify the complete path on the URL to reference the macro:

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/basemac6.file/path1.mbr/path1
```

Example 2

The INCLUDE_PATH is used in conjunction with the %INCLUDE statement, which allows you to include files. Typically, one would specify the %INCLUDE statements as follows:

```
%INCLUDE "/QSYS.LIB/NETDATA.LIB/BASEMAC3.FILE/FOOTER.MBR"
```

However, if we had an INI file that included the INCLUDE_PATH statement:

```
INCLUDE_PATH    /QSYS.LIB/NETDATA.LIB/BASEMAC3.FILE
```

then the %INCLUDE statement could have been written as follows:

```
%INCLUDE "FOOTER.MBR"
```

Example 3

The EXEC_PATH is similar to the INCLUDE_PATH, but affects references to objects specified using the %EXEC clause. For example, if program MYPROGRAM was located in library NET.DATA, then anytime you reference the following variable in a macro:

```
%define PGMVAR = %EXEC "/QSYS.LIB/NETDATA.LIB/MYPROGRAM.PGM"
```

The program MYPROGRAM would get invoked. If we had an INI file that included the EXEC_PATH statement:

```
EXEC_PATH   /QSYS.LIB/NETDATA.LIB
```

Then the %DEFINE statement could have been written as follows:

```
%define PGMVAR = %EXEC "MYPROGRAM.PGM"
```

Language environment: DTW_SQL

Using default report processing

Default report processing occurs when:

- a user-defined function is called,
- there is an Net.Data table that can be processed,
- the variable DTW_DEFAULT_REPORT is not defined, or if it is defined is set to a value of "YES", and
- there is no **%REPORT block** statement in the function.

The examples listed below assume that a SQL table, STAFFINF, exists in collection NETDATA, and has the following columns and field types: PROJNO (CHAR 6), PROJNAME (CHAR 30), DEPTNO (CHAR 3), DEPTMGR (SMALLINT), and PRSTAFF (SMALLINT). An example of what the table looks like is given below:

PROJNO	PROJNAME	DEPTNO	DEPTMGR	PRSTAFF
MA2100	MFG AUTOMATION	D11	60	12
MA2110	MFG PROGRAMMING	E21	100	3
MA2112	ROBOT DESIGN	E01	50	3
MA2113	PROD CONTROL PROG	D11	60	3
...	...	...	...	...

Example 1

The following example will result in a two-dimensional table being displayed on the browser, where rows are delimited by the dash character and columns are delimited by the vertical bar character.

```
------------------------------------------------------------
%define DATABASE = "*LOCAL"

%FUNCTION(DTW_SQL) sql1 () {
  select * from NETDATA.STAFFINF
%}

%HTML(default1) {
@sql1()
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SQLMAC1, member DEFAULT1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sqlmac1.file/default1.mbr/default1
```

Example 2

Here is the same example as above, except for the fact that the statement DTW_HTML_TABLE="YES" has been added so that the generated table is delimited using HTML table tags.

```
------------------------------------------------------------
%define DATABASE = "*LOCAL"
%define DTW_HTML_TABLE="YES"

%FUNCTION(DTW_SQL) sql1 () {
  select * from NETDATA.STAFFINF
%}

%HTML(default2) {
@sql1()
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SQLMAC1, member DEFAULT2, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sqlmac1.file/default2.mbr/default2
```

Example 3

And of course, to truely generate dynamic Web pages, you would perform the SQL SELECT statement using user input, where the data may come in as URL data or form data. In the example below, the variable reference in the SELECT statement, $(staffno), references data that came in via the URL. Note that we also specified a **%MESSAGE block** so that a message gets displayed when no records match the selection criteria.

```
------------------------------------------------------------
%define DATABASE = "*LOCAL"
%define DTW_HTML_TABLE="YES"

%MESSAGE {
  100: "No entries found that matched selection criteria.": continue
%}

%FUNCTION(DTW_SQL) sql1 () {
  select * from NETDATA.STAFFINF where prstaff=$(staffno)
%}
```

```
%HTML(default3) {
@sql1()
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SQLMAC1, member DEFAULT3, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
          netdata.lib/sqlmac1.file/default3.mbr/default3?staffno=3
```

Using user-defined report processing

User-defined report processing occurs when:

- a user-defined function is called,
- there is an Net.Data table that can be processed, and
- there is a **%REPORT block** statement in the function.

The thing to note about the %REPORT block is that that there are three areas that Net.Data processes: the area between the %REPORT block and the beginning of the %ROW block, called the **header**, which is processed one time by Net.Data; the area within the %ROW block, which is processed N times, where N is the number of rows in the table; and the area between the end of the %ROW block and the end of the %REPORT block, called the **footer**, which is processed one time by Net.Data.

When would you want to use user-defined report processing? When you want control on how the data is to be returned to the browser.

The examples listed below assume that a SQL table, STAFFINF, exists in collection NETDATA, and has the following columns and field types: PROJNO (CHAR 6), PROJNAME (CHAR 30), DEPTNO (CHAR 3), DEPTMGR (SMALLINT), and PRSTAFF (SMALLINT). An example of what the table looks like is given below:

PROJNO	PROJNAME	DEPTNO	DEPTMGR	PRSTAFF
MA2100	MFG AUTOMATION	D11	60	12
MA2110	MFG PROGRAMMING	E21	100	3
MA2112	ROBOT DESIGN	E01	50	3
MA2113	PROD CONTROL PROG	D11	60	3
...	...	...	...	...

Example 1

The following example illustrates the use of user-defined report processing. Notice the references to variables that can only be accessed within the %REPORT or %ROW blocks. In addition, the DTW_rHTMLENCODE() Net.Data built-in function is being used so that if the field data contains HTML special characters (e.g. <, >), the characters get encoded using character references in order for the characters to get displayed as ordinary text characters. The resultant data will get displayed with HTML table tags, similar to what you get in default report processing with DTW_HTML_TABLE set to "YES", except in this case we explicitly specify the HTML table tags.

```
%define DATABASE = "*LOCAL"

%FUNCTION(DTW_SQL) sql1 () {
  select * from NETDATA.STAFFINF

  %REPORT {


    $(N1)          $(N2)          $(N3)          $(N4)          $(N5)

    @dtw_      @dtw_          @dtw_          @dtw_          @dtw_
rHTMLENCODE(V1)rHTMLENCODE(V2)rHTMLENCODE(V3)rHTMLENCODE(V4)rHTMLENCODE(V5)

  %}
%}

%HTML(userdef1) {
@sql1()
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SQLMAC2, member USERDEF1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sqlmac2.file/userdef1.mbr/userdef1
```

Example 2

Here is the same query as the example above, except for the fact that we do not use table tags to display the data.

```
------------------------------------------------------------
%define DATABASE = "*LOCAL"

%FUNCTION(DTW_SQL) sql1 () {
  select * from NETDATA.STAFFINF

  %REPORT {
      %ROW {
$(N1) : @dtw_rHTMLENCODE(V1)
$(N2) : @dtw_rHTMLENCODE(V2)
$(N3) : @dtw_rHTMLENCODE(V3)
$(N4) : @dtw_rHTMLENCODE(V4)
$(N5) : @dtw_rHTMLENCODE(V5)
      %}
  %}
%}
```

```
%HTML(userdef2) {
@sql1()
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SQLMAC1, member USERDEF2, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sqlmac1.file/userdef2.mbr/userdef2
```

Example 3

Here is an example where we do not show all the data in the table on one Web page. In this example, we show the PROJNO and PROJNAME columns, and we make the data fields under the PROJNO column a link to a "detail" page, where the entire record gets displayed. We display the detail page by invoking Net.Data using the same macro, but a different HTML block. The data to be displayed is passed to Net.Data as URL data. Note that the Net.Data built-in function DTW_URLESCSEQ() is used to encode characters that are not allowed in URLs.

```
-------------------------------------------------------------
%define {
 DATABASE = "*LOCAL"
 PREF="/CGI-BIN/DB2WWW/QSYS.LIB/NETDATA.LIB/SM2.FILE/UD3.MBR/detail"
%}

%FUNCTION(DTW_SQL) sql1 () {
  select * from NETDATA.STAFFINF

  %REPORT {


                                                %ROW { @dtw_URLESCSEQ(V1, f1)
                                                @dtw_URLESCSEQ(V2, f2)
        $(N1)          $(N2)                    @dtw_URLESCSEQ(V3, f3)
                                                @dtw_URLESCSEQ(V4, f4)
                                                @dtw_URLESCSEQ(V5, f5)
```

@dtw_rHTMLENCODE(V1)@dtw_rHTMLENCODE(V2)%}

```
  %}
%}

%HTML(userdef3) {
@sql1()
%}
%HTML(detail) {
PROJNO : @dtw_rHTMLENCODE(pn)
```

```
PROJNAME : @dtw_rHTMLENCODE(pnm)
DEPTNO : @dtw_rHTMLENCODE(dept)
DEPTMGR : @dtw_rHTMLENCODE(deptm)
PRSTAFF : @dtw_rHTMLENCODE(pst)
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SM2, member UD3, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sm2.file/ud3.mbr/userdef3
```

Example 4

This example illustrates the way one would save data obtained from a call to a function for use outside of the function block. In this case, the HTML section references global variables that are set to values after a call to a function. We assume that user input comes in as URL data that results in a query match of either one record or no record. If there is no record match, the **%MESSAGE block** will get control, issue a message indicating the information was not found, and exit macro processing.

```
-------------------------------------------------------------
%define {
 DATABASE = "*LOCAL"

 my_projno = ""
 my_projname = ""
 my_deptno = ""
 my_deptmgr = ""
 my_prstaff = ""

%}

%FUNCTION(DTW_SQL) sql1 () {
  select * from NETDATA.STAFFINF where projno=$(inprojno)

  %REPORT {
      %ROW {
        @dtw_HTMLENCODE(V1, my_projno)
        @dtw_HTMLENCODE(V2, my_projname)
        @dtw_HTMLENCODE(V3, my_deptno)
        @dtw_HTMLENCODE(V4, my_deptmgr)
        @dtw_HTMLENCODE(V5, my_prstaff)
      %}
  %}

  %MESSAGE {
    100: "Sorry, no information found that matched selection criteria.": exit
  %}
```

```
%}

%HTML(userdef4) {
@sql1()

This is a paragraph that contains information that was
dynamically generated.  You wanted to see the information
relating to project number $(my_projno).  The project
name associated with this project number is $(my_projname) and
is managed by manager number $(my_deptmgr).
The department number that the project belongs to is
$(my_deptno), and there are $(my_prstaff)
people working on the project.
%}
-------------------------------------------------------------
```

Calling a SQL function from within a row block of another function

Calling functions from within %ROW blocks is the same as if you are calling the function from an %HTML block. However, there is one important difference. Within a %ROW block, there may be a Net.Data table variable that is being processed by Net.Data, and if you call another function that uses the same table, things can get messy. The proper way to call a function that does table processing from within the %ROW block of another function is to pass that functions being called a Net.Data table variable defined by you. This ensures that unique tables are being used within function blocks.

Example 1

The following example illustrates the use of user-defined tables that are passed to functions that processes tables. On the initial call to function sql1 we pass in a table we defined and named mytable1, and from within this function we call function sql2, passing it a different table, mytable2, in addition to a field we obtained on the first query that is used in the second query to obtain a record from the database.

```
------------------------------------------------------------
%define DATABASE = "*LOCAL"

%define mytable1 = %TABLE
%define mytable2 = %TABLE

%FUNCTION(DTW_SQL) sql2 (IN p1, OUT t2) {
  select * from NETDATA.STAFFINF where projno='$(p1)'

  %REPORT {
      %ROW {
 $(N1) is @dtw_rHTMLENCODE(V1)

 $(N2) is @dtw_rHTMLENCODE(V2)

 $(N3) is @dtw_rHTMLENCODE(V3)

 $(N4) is @dtw_rHTMLENCODE(V4)

 $(N5) is @dtw_rHTMLENCODE(V5)
      %}
  %}
%}

%FUNCTION(DTW_SQL) sql1 (OUT t1) {
  select * from NETDATA.STAFFINF

  %REPORT {
      %ROW {
        @sql2(V1, mytable2)
```

```
      %}
  %}
%}

%HTML(netcall1) {

@sql1(mytable1)
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SQLMAC2, member NESTCALL1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sqlmac2.file/netcall1.mbr/netcall1
```

Note that this is a simplistic example in that in both SQL functions the same database is queried. However, prior to call sql2(), one can use the Net.Data built-in function DTW_ASSIGN() to set the DATABASE variable to another database (i.e. a remote database), so that when sql2() is called, the query will be done on the remote system.

Enabling system naming mode

There are two naming conventions that can be used in DB2 for i: system and SQL. The naming convention used affects the method for qualifying file and table names.

In the system naming convention, files are qualified by library name in the form:

```
library/file
```

If the file is not explicitly qualified and depending on the SQL operation, either the current library (*CURLIB) or the library list (*LIBL) is used when system naming convention is enabled.

In the SQL naming convention, tables are qualified by the collection name in the form:

```
collection.table
```

If the table name is not explicitly qualified, the default qualifier is the user profile of the job running the SQL statement.

For Net.Data, the default is the SQL naming convention. If system naming convention is desired, insert the following line in the Net.Data initialization file:

```
DTW_SQL_NAMING_MODE    SYSTEM_NAMING
```

You can also set DTW_SQL_NAMING_MODE to SQL_NAMING, which is the same as the default of SQL naming convention.

Note that when a connection is made to a remote IBM i, the SQL Call Level Interface looks for a *SQLPKG object in library QGPL by the name of QSQCLIPKG. If it exists, it is used, but if it does not exist it is created.

This SQL package contains all the rules by which the native SQL is accessed. Therefore, the first connection's attributes set the rules that all subsequent connections must follow. One of these attributes is naming convention.

If you set the DTW_SQL_NAMING_MODE to conflict with the naming convention in an existing QGPL/QSQCLIPKG on a remote IBM i, the SQL statement in the Web macro results in an SQLCODE of -5016.

To avoid this situation, select a naming convention and stick with it. If the QGPL/QSQCLIPKG object exists that is in conflict with the naming convention you have chosen, delete it and issue the Net.Data request again, which will result in the creation of a new QGPL/QSQCLIPKG with the naming convention you require.

Using the START_ROW_NUM report variable

If you are not happy with the performance of Net.Data, then the use of the START_ROW_NUM report variable is a must. This variable sets the row number to begin displaying results contained in a Net.Data table. Using this variable together with RPT_MAX_ROWS, one can break large tables into smaller sets.

Example 1

The following macro is a complete implementation of next/previous functionality using START_ROW_NUM:

1. You need to replace the query in myQuery() with whatever is appropriate for you.
2. The table results are saved in a variable and passed to a REXX function which determines whether or not we are on a boundary condition for the rows being displayed. This will affect whether the "next" and "previous" buttons are to be displayed. For example, we do not want to display "next" if there are no more records to display.
3. Note that the query must be performed each time, since variables are not persistent across macro invocations.

```
------------------------------------------------------------

%define {
   START_ROW_NUM = "1"
   RPT_MAX_ROWS = "25"
   resultTable = %table
%}

%function(DTW_SQL) myQuery(OUT table) {
    select * from NETDATADEV.CUSTOMER
    %report{%}
%}

%function(DTW_REXX) displayRows(INOUT startRow, direction,
                                table, IN pageSize) {
 if (direction = 'next') then
   do
     startRow = startRow + pageSize
     if ((table_ROWS+1) - startRow < pageSize) then
       direction = 'prev_only'
     else
       direction = 'both'
   end
 else if (direction = 'previous') then
   do
     startRow = startRow - pageSize
     if (startRow = 1) then
```

```
      direction = 'next_only'
    else
      direction = 'both'
  end
 else if (direction = '') then
  do
    startRow = 1
    if (startRow + pageSize > table_ROWS) then
      direction = 'neither'
    else
      direction = 'next_only'
    end
%}

%html(report) {
@myQuery(resultTable)

<form method="post" action="report">

@displayRows(START_ROW_NUM, submit, resultTable,
             RPT_MAX_ROWS)

<input type="hidden" value="$(START_ROW_NUM)" name="START_ROW_NUM" />
%if (submit == "both" || submit == "next_only")
   <input type="submit" value="next" name="submit" />
%endif

%if (submit == "both" || submit == "prev_only")
   <input type="submit" value="previous" name="submit" />
%endif

</form>

%}

------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SQLMAC7, member STARTROW, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
          netdata.lib/sqlmac7.file/startrow.mbr/report
```

Language environment: DTW_SYSTEM

Issuing CL commands (and calling programs)

You can use the SYSTEM (DTW_SYSTEM) language environment to issue any CL command, which also includes calling programs. Note that the user profile of the HTTP server job must have authority to any command that is to be issued.

The statement to be processed must be specified within an **%EXEC block**. Any Net.Data variable references within the %EXEC block will get replaced by the actual string value prior to the issuing of the command. The syntax of the statement is similar to what you would specify interactively from the CL command line, except for a few differences that will be illustrated in the following examples. In addition, when calling programs, you should be aware of the following with regards to the parameters you pass to the program:

- Numeric constants are passed as packed decimal digits.
- Characters that are not enclosed in single quotation marks are folded to uppercase.
- Characters that are enclosed in single quotation marks are not changed (i.e. mixed case strings are supported).
- Any parameters that are passed on the call statement (statements within the %EXEC block) are considered input type parameters (i.e. the parameters passed to the program can be used and manipulated by the program, but changes to the parameters are not reflected back to Net.Data). See Retrieving and Updating Net.Data Variables to find out how programs can update Net.Data Web macro variables.

Example 1

The following example simply adds a library to the library list. The **%MESSAGE** block is used so that any errors that may occur when issuing the command are ignored.

```
------------------------------------------------------------
%FUNCTION(DTW_SYSTEM) sys1 () {
  %EXEC {
     /QSYS.LIB/ADDLIBLE.CMD MYLIB
  %}

  %MESSAGE {
     default: "" : continue
  %}
%}

%HTML(cmd1) {

Adding library list entry
@sys1()
```

```
After which you can perform some functions that may use objects in the
library that was added to the library list.
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SYSMAC1, member CMD1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sysmac1.file/cmd1.mbr/cmd1
```

Example 2

The following example sends a break message to all workstations on the system via the SNDBRKMSG CL command. The message itself comes in as URL data from the browser (it typically would come in as form data, to allow the user to type in the message, but URL data is a convenient way to illustrate the point).

```
-------------------------------------------------------------
%FUNCTION(DTW_SYSTEM) sys1 () {
  %EXEC {
     /QSYS.LIB/SNDBRKMSG.CMD MSG('$(msg)') TOMSGQ(*ALLWS)
  %}
%}

%HTML(cmd2) {

Sending break message to all workstations.  Message that will be sent is:

DTW_rHTMLENCODE(msg)
@sys1()
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SYSMAC1, member CMD2, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sysmac1.file/cmd2.mbr/cmd2?msg=Test%20Message
```

Example 3

Here is an example of calling a program. No parameters are passed to the program.

```
-------------------------------------------------------------
%FUNCTION(DTW_SYSTEM) sys1 () {
  %EXEC {
     /QSYS.LIB/NETDATA.LIB/CMDPGM.PGM
  %}
%}
```

```
%HTML(cmd3) {

Calling a program with no parameters.
@sys1()
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SYSMAC1, member CMD3, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/sysmac1.file/cmd3.mbr/cmd3
```

Note that the statement in the %EXEC block in the above macro is equivalent to the following statement:

```
------------------------------------------------------------
  %EXEC {
     /QSYS.LIB/CALL.CMD NETDATA/CMDPGM
  %}
------------------------------------------------------------
```

Example 4

Here is an example of calling a program and passing it parameters. Note that the variable reference in the %EXEC block will get substituted with the variable value. In this case, the variable referenced came in on the URL request. One other thing to note about the parameters being passed is that the 99 that is passed to the program is passed to the program as a packed decimal number.

```
------------------------------------------------------------
%FUNCTION(DTW_SYSTEM) sys1 () {
  %EXEC {
     /QSYS.LIB/NETDATA.LIB/CMDPGMP.PGM (astring
                                        99
                                        '$(INDATA1)')
  %}
%}

%HTML(cmd4) {

Calling a program with parameters.
@sys1()
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SYSMAC1, member CMD4, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
         netdata.lib/sysmac1.file/cmd4.mbr/cmd4?INDATA1=Parm3
```

Retrieving and updating Net.Data variables

You should understand all about Net.Data variables by going over the examples in "Defining, Referencing, and Setting Variables". If you have already done so then you are ready to learn the answer to the age-old question: "How the heck can a program that gets called update a Net.Data variable?".

There are two ways to pass information to a program that is invoked by the SYSTEM (DTW_SYSTEM) language environment:

1. Directly, by passing parameters to a program as was shown in Issuing CL Commands (and Calling Programs). The parameters that are passed to the program are considered input type parameters (i.e. the parameters passed to the program can be used and manipulated by the program, but changes to the parameters are not reflected back to Net.Data).
2. Indirectly, via **environment variables**. Environment variables are character strings of the form "name=value" that are stored in an environment space outside of the program. The strings are stored in a temporary space associated with the job.

   When a DTW_SYSTEM language environment function is called, any function parameters that are input (IN) or input/output (INOUT) are stored in the environment space by the DTW_SYSTEM language environment prior to executing the statement within the **%EXEC block**. Upon the successful completion of the statement, the DTW_SYSTEM language environment determines whether there are any output (OUT or INOUT) function parameters, and if so, it retrieves the value corresponding to the function parameter from the environment space and updates the function parameter value with the new value. When Net.Data gets control, it in turn updates all OUT or INOUT parameters with the new values obtained from the DTW_SYSTEM language environment.

Environment variables can be set and retrieved using the following APIs:

ILE Programming language	To retrieve, use	To set, use
C, C++	getenv()	putenv()
CL(1), RPG, COBOL	QtmhGetEnv()(2)	QtmhPutEnv()(3)

1. For V3R7 and on, one can also use the CHGENVVAR and ADDENVVAR CL commands to set an environment variable.
2. QtmhGetEnv() is shipped as part of IBM TCP/IP Connectivity Utilities/400.
3. QtmhPutEnv() was not originally shipped as part of IBM TCP/IP Connectivity Utilities/400 for V3R2 and V3R7. It was added later in the cycle and can be obtained via the V3R2 PTF 5763TC1-SF40953 or the V3R7 PTF 5716TC1-SF40954.

This method is the only way that a program can update Net.Data variables and have the update reflected back to Net.Data.

Confused? The following example should help clarify things.

Example 1

This example includes a macro that contains a function definition with 3 parameters, P1, P2, and P3. P1 is an input (IN) parameter and P2 and P3 are output (OUT) parameters. The function invokes a program. The program name is obtained from the URL. The program will update P2 with the value of P1 and set P3 to a character string. Prior to processing the statement in the %EXEC block, the DTW_SYSTEM language environment will store P1 and the corresponding value in the environment space.

```
------------------------------------------------------------
%DEFINE {
  MYPARM2 = "ValueOfParm2"
  MYPARM3 = "ValueOfParm3"
%}

%FUNCTION(DTW_SYSTEM) sys1 (IN P1, OUT P2, P3) {
  %EXEC {
     /QSYS.LIB/NETDATA.LIB/$(PGMNAME).PGM
  %}
%}

%HTML(upd1) {

Passing data to a program.  The current value
of MYPARM2 is "$(MYPARM2)", and the current value of MYPARM3 is
"$(MYPARM3)".  Now we invoke the Web macro function.

@sys1("ValueOfParm1", MYPARM2, MYPARM3)

After the function call, the value of MYPARM2 is "$(MYPARM2)",
and the value of MYPARM3 is "$(MYPARM3)".
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file SYSMAC2, member UPD1, we reference the macro by loading the following URL (from a browser):

```
http://hostname/cgi-bin/db2www/qsys.lib/
       netdata.lib/sysmac2.file/upd1.mbr/upd1?PGMNAME=UPDRPG1
```

Note that we specified as URL data the RPG program. We can obtain the same results by specifying the C program, UPDC1, or the CL program, UPDCL1.

Here is UPDC1 program written in the **C programming language**:

```
------------------------------------------------------------
#include <stdlib.h>
#include <stdio.h>
```

```
#include <qp0z1170.h>

int main()
{
    char *p1_value;
    char env_buffer[100];

    /* Get P1 value */
    p1_value = getenv("P1");

    /* Set P2 to value of P1 */
    sprintf(env_buffer, "%s=%s", "P2", p1_value);
    putenv(env_buffer);

    /* Set P3 to a new value */
    putenv("P3=newValue");
    return 0;
}
------------------------------------------------------------
```

The C program was created by issuing the CRTBNDC CL command as follows:

```
CRTBNDC PGM(NETDATA/UPDC1) SRCFILE(NETDATA/QCSRC)
```

Here is UPDRPG1 program written in the **RPG programming language**:

```
------------------------------------------------------------
 * prototype for QtmhGetEnv API
Dgetenv           PR                extproc('QtmhGetEnv')
D                         32767
D                          10i 0
D                          10i 0
D                          30
D                          10i 0
D                          16
 * prototype for QtmhPutEnv API
Dputenv           PR                extproc('QtmhPutEnv')
D                         32767
D                          10i 0
D                          16
 * variables used for QtmhGetEnv & QtmhPutEnv calls
DenvVal           S         32767
DenvBufLn         S           10i 0 inz(%size(envVal))
DenvValLn         S           10i 0
DenvNm            S           30
DenvNmLn          S           10i 0
 * Error data structure from openness library, QSYSINC.
 * If not present, it can be installed (it comes with
```

```
 * OS/400 as part of the System Openness Includes component).
 /copy qsysinc/qrpglesrc,qusec
 * Initialize the error data structure's bytes-provided subfield
C              eval      qusbprv = 16                 qusec length
 * Get P1 value
C              eval      envNm = 'P1'
C     ' '      checkr    envNm         envNmLn
C              callp     getenv(envVal:envBufLn:envValLn:
C                                envNm:envNmLn:qusec)
 * Set P2 to value of P1
C              eval      envVal = 'P2=' + %subst(envVal:1:envValLn)
C     ' '      checkr    envVal        envValLn
C              callp     putenv(envVal:envValLn:qusec)
 * Set P3 to a new value
C              eval      envVal = 'P3=newValue'
C     ' '      checkr    envVal        envValLn
C              callp     putenv(envVal:envValLn:qusec)
C              eval      *inlr = *on
------------------------------------------------------------
```

The RPG program was created by issuing the following CL commands:

```
CRTRPGMOD MODULE(NETDATA/UPDRPG1) SRCFILE(NETDATA/QRPGSRC)
CRTPGM PGM(NETDATA/UPDRPG1) MODULE(NETDATA/UPDRPG1)
    BNDSRVPGM(QTCP/QTMHCGI)
```

Here is UPDCL1 program written in the **CL programming language**:

```
------------------------------------------------------------
PGM
DCL        VAR(&ENVVAL) TYPE(*CHAR) LEN(100)
DCL        VAR(&ENVVALLN) TYPE(*CHAR) LEN(4)
DCL        VAR(&ERRCODE) TYPE(*CHAR) LEN(16)

DCL        VAR(&ENV_BUFFER) TYPE(*CHAR) LEN(100)
DCL        VAR(&ENVBUFLN) TYPE(*CHAR) LEN(4)
DCL        VAR(&DECVAL) TYPE(*DEC)

/* Get P1 value */
CALLPRC    PRC('QtmhGetEnv') PARM(&ENVVAL X'00000064' +
                   &ENVVALLN 'P1' X'00000002' +
                   &ERRCODE)

/* Set P2 to value of P1 */
CHGVAR  VAR(&DECVAL) VALUE(%BIN(&ENVVALLN 1 4))
CHGVAR  VAR(&ENV_BUFFER) VALUE('P2=')
CHGVAR  VAR(%SST(&ENV_BUFFER 4 &DECVAL)) VALUE(&ENVVAL)
CHGVAR  VAR(&DECVAL) VALUE(&DECVAL+3)
CHGVAR  VAR(%BIN(&ENVBUFLN)) VALUE(&DECVAL)
```

```
CALLPRC PRC('QtmhPutEnv') PARM(&ENV_BUFFER &ENVBUFLN &ERRCODE)

/* Set P3 to a new value */
CALLPRC PRC('QtmhPutEnv') PARM('P3=newValue' X'0000000B' &ERRCODE)
ENDPGM
------------------------------------------------------------
```

The CL program was created by issuing the following CL commands:

```
CRTCLMOD MODULE(NETDATA/UPDCL1) SRCFILE(NETDATA/QCLSRC)
CRTPGM PGM(NETDATA/UPDCL1) MODULE(NETDATA/UPDCL1)
    BNDSRVPGM(QTCP/QTMHCGI)
```

Language environment: DTW_REXX

Running REXX programs

The REXX (DTW_REXX) language environment can interpret in-line REXX programs which are specified in a **%FUNCTION block** of the Web macro, or it can execute external REXX programs (specified within the **%EXEC block**) stored in a separate file. External REXX programs provide a slight performance benefit over in-line REXX programs. Any Net.Data variable references within the %EXEC block or within the in-line REXX statements will get replaced by the actual string value prior to the processing of the statement(s).

Example 1

The following example calls an external REXX program, passing the REXX program an argument string (a reference to a Net.Data variable that came in as URL data) that it can reference via the REXX **PARSE ARG** instruction.

```
------------------------------------------------------------
%function(DTW_REXX) rexx1() {
  %EXEC{
     /QSYS.LIB/NETDATA.LIB/QREXXSRC.FILE/CALL1.MBR $(INPARM1)
  %}
%}

%HTML(call1) {

Calling an external REXX program
@rexx1()
%}
------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file REXMAC1, member CALL1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
       netdata.lib/rexmac1.file/call1.mbr/call1?INPARM1=APPLE
```

Example 2

The following example has REXX coded within the macro. The REXX statement is simply a "return" statement.

```
------------------------------------------------------------
%FUNCTION(DTW_REXX) rexx1 () {
   return
%}

%HTML(call2) {
```

```
Example of using in-line REXX
@rexx1()
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file REXMAC1, member CALL2, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/rexmac1.file/call2.mbr/call2
```

Retrieving and updating Net.Data variables

You should understand all about Net.Data variables by going over the examples in "Defining, Referencing, and Setting Variables".

There are two ways to pass information to a REXX program that is invoked by the REXX (DTW_REXX) language environment:

1. Directly, by passing parameters to an external REXX program as was shown in "Running REXX Programs". The parameters that are passed to the program are considered input type parameters (i.e. the parameters passed to the program can be used and manipulated by the program, but changes to the parameters are not reflected back to Net.Data).
2. Indirectly, via the REXX program **variable pool**. Whenever a REXX program is started, a space which contains information about all variables is created and maintained by the REXX interpreter. This space is called the variable pool. There are interfaces to access the variable pool, which the DTW_REXX language environment uses to set and retrieve variables.

   When a DTW_REXX language environment function is called, any function parameters that are input (IN) or input/output (INOUT) are stored in the variable pool by the DTW_REXX language environment prior to executing the REXX program. When the REXX program is invoked, it can access these variables directly. Upon the successful completion of the REXX program, the DTW_REXX language environment determines whether there are any output (OUT) or INOUT function parameters, and if so, it retrieves the value corresponding to the function parameter from the variable pool and updates the function parameter value with the new value. When Net.Data gets control, it in turn updates all OUT or INOUT parameters with the new values obtained from the DTW_REXX language environment.

Example 1

This example includes a macro that contains a function definition with 3 parameters, P1, P2, and P3. P1 is an input (IN) parameter and P2 and P3 are output (OUT) parameters. The DTW_REXX language environment processes the in-line REXX program. The REXX program will update P2 with the value of P1 and set P3 to a character string. Prior to processing the statement in the %EXEC block, the DTW_SYSTEM language environment will store P1 and the corresponding value in the variable pool.

```
------------------------------------------------------------
%DEFINE {
  MYPARM2 = "ValueOfParm2"
  MYPARM3 = "ValueOfParm3"
%}

%FUNCTION(DTW_REXX) rexx1 (IN P1, OUT P2, P3) {

  /* Following are in-line REXX statements */
  P2 = P1
  P3 = 'newValue'
  return 0
```

```
%}

%HTML(upd1) {

Passing data to a REXX program.  The current value
of MYPARM2 is "$(MYPARM2)", and the current value of MYPARM3 is
"$(MYPARM3)".  Now we invoke the Web macro function.

@rexx1("ValueOfParm1", MYPARM2, MYPARM3)

After the function call, the value of MYPARM2 is "$(MYPARM2)",
and the value of MYPARM3 is "$(MYPARM3)".
%}
-------------------------------------------------------------
```

Assuming that the Web macro is stored in library NETDATA, file REXMAC2, member UPD1, we reference the macro by loading the following URL (from a browser):

```
   http://hostname/cgi-bin/db2www/qsys.lib/
            netdata.lib/rexmac2.file/upd1.mbr/upd1
```