Net.Data Reference

Net.Data Reference

Note

Be sure to read the information in “Appendix C. Notices” on page 231 before using this information and the product it supports.

Fourth Edition (June 1998)

Contents

Preface

Thank you for selecting Net.Data Version 2, IBM's development tools for creating dynamic Web pages! With Net.Data you can rapidly develop Web pages with a dynamic content by incorporating data from a variety of data sources and by using the power of programming languages you already know.

Net.Data Version 2 provides significantly improved performance along with new features that give you the power to build and deploy your Internet business solutions.

About Net.Data

With IBM's Net.Data product, you can create dynamic Web pages using data from both relational and non-relational database management systems (DBMSs), including DB2, IMS, and ODBC-enabled databases, and using applications written in programming languages such as Java, JavaScript, Perl, C, C++, and REXX.

You can think of Net.Data as a macro processor that executes as middleware on a Web server. You can write Net.Data application programs, called macros, that Net.Data interprets to create dynamic Web pages with customized content based on input from the user, the current state of your databases, existing business logic, and other factors that you design into your macro.

A request, in the form of a URL (uniform resource locator), flows from a browser, such as Netscape or Internet Explorer, to a Web server that forwards the request to Net.Data for execution. Net.Data locates and executes the macro, and builds a Web page that it customizes based on functions that you write. These functions can:

- Encapsulate business logic within Perl scripts, C and C++ applications, or REXX programs
- Access databases such as DB2

Net.Data supports industry-standard interfaces such as HyperText Transfer Protocol (HTTP) and Common Gateway Interface (CGI). HTTP is used between the browser and the Web server, and CGI is used between the Web server and Net.Data. This lets you select your favorite browser or web server for use with Net.Data. Net.Data also supports FastCGI and the major Web server APIs on multiple operating systems.

About This Book

This book explains the syntax and usage of Net.Data language constructs, variables, and functions in general.

This book might refer to products or features that are announced, but not yet available.

More information, sample Net.Data macros, demos, and the latest copy of this book, is available from the following World Wide Web sites:

- http://www.software.ibm.com/data/net.data
- http://www.as400.ibm.com/netdata

Who Should Read This Book

People involved in planning and writing Net.Data applications can use the information in this book to understand what language constructs, variable, and functions Net.Data provides.

To understand the concepts discussed in this book, you should be familiar with Web servers, simple SQL statements, and HTML (including using HTML forms), and the information in *Net.Data Administration and Programming Guide* and *Net.Data Language Environment Reference*.

About Examples in This Book

Examples used in this book are kept simple to illustrate specific concepts and do not show every way Net.Data constructs can be used. Some examples are fragments that do not work alone.

How to Read the Syntax Diagrams

The following rules apply to the syntax diagrams used in this book:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a statement.

  The ───► symbol indicates that the statement syntax is continued on the next line.

  The ►── symbol indicates that a statement is continued from the previous line.

  The ───►◄ symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ►── symbol and end with the ───► symbol.
- Required items appear on the horizontal line (the main path).

```
►►──required_item────────────────────────────────────────────►◄
```

- Optional items appear below the main path.

```
►►──required_item──┬───────────────┬─────────────────────────►◄
                   └─optional_item─┘
```

  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

```
                   ┌─optional_item─┐
►►──required_item──┴───────────────┴─────────────────────────►◄
```

- If you can choose from two or more items, they appear vertically, in a stack.

  If you *must* choose one of the items, one item of the stack appears on the main path.

```
►►──required_item──┬─required_choice1─┬────────────────────────────────►◄
                   └─required_choice2─┘
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
►►──required_item──┬──────────────────┬────────────────────────────────►◄
                   ├─optional_choice1─┤
                   └─optional_choice2─┘
```

If one of the items is the default, it appears above the main path and the remaining choices are shown below.

```
                   ┌─default_choice──┐
►►──required_item──┼─────────────────┼─────────────────────────────────►◄
                   ├─optional_choice─┤
                   └─optional_choice─┘
```

- An arrow returning to the left, above the main line, indicates an item that can be repeated.

```
                   ┌────────────────────┐
                   ▼                    │
►►──required_item────repeatable_item────┴──────────────────────────────►◄
```

If the repeat arrow contains punctuation, you must separate repeated items with the specified punctuation.

```
                   ┌─,──────────────────┐
                   ▼                    │
►►──required_item────repeatable_item────┴──────────────────────────────►◄
```

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Keywords appear in uppercase (for example, `FROM`). In Net.Data, keywords can be in any case. Terms that are not keywords appear in lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Chapter 1. Net.Data Macro Language Constructs

This chapter describes the Net.Data macro syntax and the language constructs used in the Net.Data macro file. The language constructs consist of a keyword and a statement or block in the Net.Data macro, specify different variable types, and perform other special tasks such as including files.

This chapter describes:

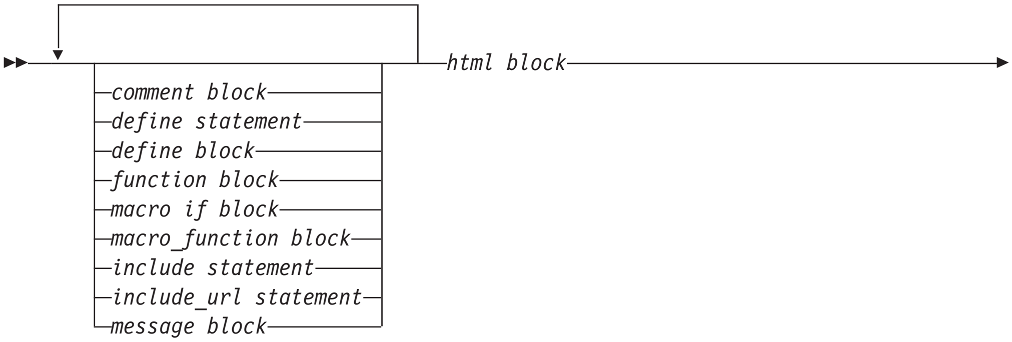
- "Net.Data Macro File Syntax"
- "Common Syntax Elements" on page 4
- "Macro Language Constructs" on page 5

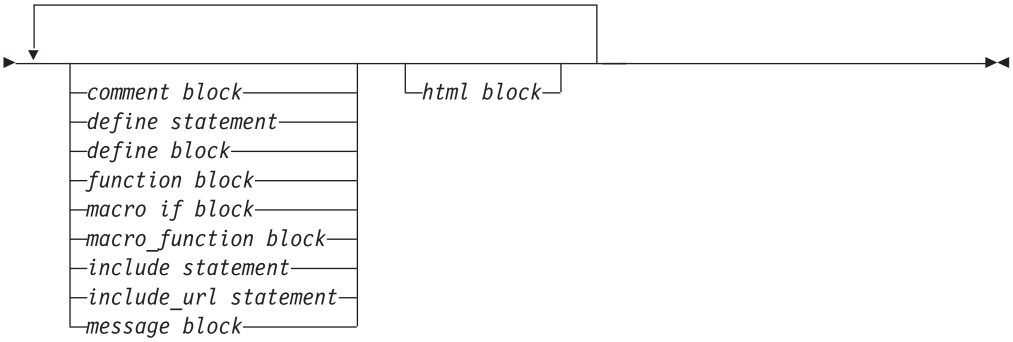
Net.Data Macro File Syntax

A Net.Data macro is a plain text file consisting of a series of Net.Data macro language constructs that:

- Specify the layout of Web pages
- Define variables and functions
- Call functions that are defined in the macro file or that Net.Data passes to language environments for processing
- Format the processing output in HTML and return it to the Web browser

Each statement is composed of one or more language constructs, which in turn are composed of keywords, special characters, strings, names, and variables. The following diagram depicts the global structure of a syntactically valid Net.Data macro. See "Chapter 1. Net.Data Macro Language Constructs" for detailed syntax of each element in the global structure.





The Net.Data macro contains two parts: the declaration part and the HTML part. You can use these parts multiple times and in any order.

- *Declaration part* contains the definitions of variables and functions in the macro file.
- *HTML part* contains HTML blocks that contain HTML statements that specify the layout of the Web page. This part includes the report section.

Figure 1 shows the declaration and HTML parts of the macro file.

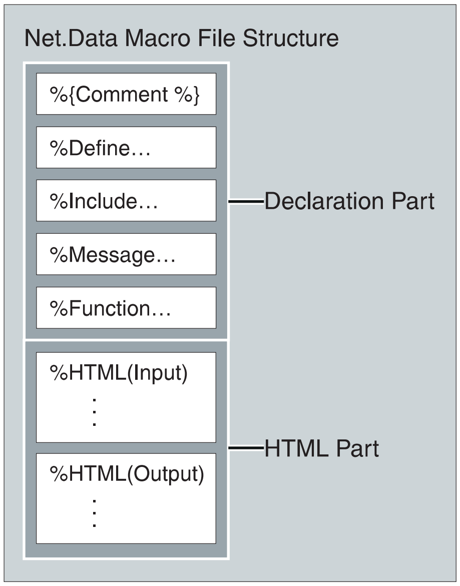


Figure 1. Macro File Structure

Variables and functions that are used in the declaration or HTML part must be defined before being used by a variable reference or a function call.

Figure 2 on page 3 demonstrates the parts of a macro file. The declaration part contains the DEFINE and FUNCTION definition blocks. The HTML blocks act as input and output blocks.

```
%{ **********************       Define block        ************************%}
%DEFINE {
   page_title="Net.Data macro Template"
%}

%{ ********************** Function Definition block ************************%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result)
    { %EXEC{ompsamp.cmd %}
%}

%FUNCTION(DTW_REXX) today () RETURNS(result)
  {
      result = date()
%}

%{ **********************      HTML Block: Input    ************************%}
%HTML (INPUT) {
<html>
<head>
<title>$(page_title)<title>
</head><body>
<h1>Input Form</h1>
Today is @today()

<FORM  METHOD="post" ACTION="output">
Type some data to pass to a REXX program:
<INPUT NAME="input_data" TYPE="text" SIZE="30">
<p>
<INPUT TYPE="submit" VALUE="Enter">

<hr>
<p>[<a href="/">Home page]
</body></html>
%}

%{ **********************    HTML Block: Output     ************************%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title>
</head><body>
<h1>Output Page</h1>
<p>@rexx1(input_data)
<p><hr>
<p>[<a href="/">Home page</a> |
<a href="input">Previous page</a>]
</body></html>
%}
```

Figure 2. The Macro File Template Format

The Net.Data macro language is a free-form language, giving you flexibility for writing your macros. Unless specifically noted, extra white space characters are ignored. Each of the Net.Data macro language constructs is described in the following section, along with several other elements that are used to define the constructs. The Net.Data macro language supports DB2 WWW Connection language elements for backward compatibility. Although these language elements are described in “Appendix A. DB2 WWW Connection” on page 223, it is recommended that you use the Net.Data language constructs.

The examples show some of the ways you can use the language constructs, variables, functions, and other elements in your macro files. You can download the samples and demos from the Net.Data Web pages for more extensive examples:

- http://www.software.ibm.com/data/net.data
- http://www.as400.ibm.com/net.data

Common Syntax Elements

The following syntax elements are used frequently in the language construct descriptions:

- “Variable Name”
- “Variable Reference”
- “Strings” on page 5

Variable Name

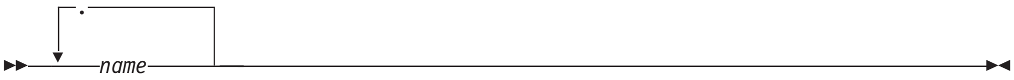
Purpose:

Identifies one or more names; each subsequent name is concatenated by a period (.). A name is an alphabetic or numeric string beginning with an alphabetic character or underscore and containing any combination of alphabetic, numeric, or underscore characters.

Strings in quotes (“”), can contain any character except the new-line character. If the string is in brackets, ({ %}), it can contain any character including the new-line character.

Variable names must begin with a letter or underscore (_) and contain any alphanumeric characters or underscore. All variable names are case sensitive except N_*columnName* and V_*columnName* (See “Net.Data Table Processing Variables” on page 63 for more information about these two exceptions.).

Syntax:



Variable Reference

Purpose:

Returns the value of a previously defined variable and is specified with $ and (). For example: if VAR = ’abc’, $(VAR) returns the value ’abc’. Variable references are evaluated during run time. When a variable is defined for an EXEC statement or block, Net.Data runs the specified action when it reads the variable reference.

The variable that is referenced must be defined in the Net.Data macro before being referenced. If the variable is not defined, an empty string is returned.

Syntax:

```
►►—$—(—variable_name—)——————————————————————————————————►◄
```

Strings

Any sequence of alphabetic and numeric characters and punctuation. If the string appears within double quotes, the new-line character is not allowed. See the string parameter description in each language construct for restrictions when used with the language construct.

Two pairs of double quotes ("""") inside a string are treated as one pair of double quotes (""). To specify that a string contains double quotes, use two pairs of double quotes. For example, if you define a string value as:

```
%DEFINE result = " ""Hello world!"" "
```

The value of *result* is:

```
"Hello world!"
```

A string used as function argument or as term in a comparison expression can contain pairs of double quotes.

An HTML statement is a string.

Macro Language Constructs

This section describes the language constructs used in the Net.Data macro file.

Each language construct description can contain the following information:

Purpose
Defines why you use the language construct in the Net.Data macro.

Syntax
Provides a diagram of the language construct's logical structure.

Parameters
Defines all the elements in the syntax diagram and provides cross references to other language constructs' syntax and examples.

Context
Explains where in the Net.Data macro structure the language construct can be used.

Restrictions
Defines which elements it can contain and specifies any usage restrictions.

Examples
Provides simple examples and explanations for using the keyword statement or block within the Net.Data macro.

The following constructs are used in the macro; please refer to each constructs description for syntax and examples.

- "Comment Block" on page 7
- "DEFINE Block or Statement" on page 9
- "ENVVAR Statement" on page 13

Comment Block

Purpose

Documents the functions of the Net.Data macro. Because the COMMENT block can be used anywhere in the macro file, it is not documented in the other syntax diagrams.

Syntax

```
►►—%{—text—%}———————————————————————————————————————————————————————►◄
```

Values

text	Any string on one or more lines. Net.Data ignores the contents of all comments.

Context

Comments can be placed anywhere between Net.Data language constructs in a Net.Data macro.

Restrictions

Any text or characters are allowed; however, comment blocks cannot be nested.

Examples

Example 1: A basic comment block

```
%{
This is a comment block. It can contain any number of lines
and contain any characters. Its contents are ignored by Net.Data.
%}
```

Example 2: Comments in a FUNCTION block

```
%function(DTW_REXX) getAddress(IN  name,   %{ customer name %}
                               IN  phone,  %{ customer phone number %}
                               OUT address %{ customer address %}
                              )
{
     ....
%}
```

Example 3: Comments in an HTML block

```
%html(report) {

%{ run the query and save results in a table %}
@myQuery(resultTable)

%{ build a form to display a page of data %}
<form method="POST" action="report">

%{ send the table to a REXX function to send the data output %}
@displayRows(START_ROW_NUM, submit, resultTable, RPT_MAX_ROWS)

%{ pass START_ROW_NUM as a hidden variable to the next invocation %}
<input name="START_ROW_NUM" type="hidden" value="$(START_ROW_NUM)">

%{ build the next and previous buttons %}
%if (submit == "both" || submit == "next_only")
  <input name="submit" type="submit" value="next">
  %endif
%if (submit == "both" || submit == "prev_only")
  <input name="submit" type="submit" value="previous">
  %endif
</form>
%}
```

Example 4: Comments in a DEFINE block

```
%define {
   START_ROW_NUM = "1"        %{ starting row number for output table %}
   RPT_MAX_ROWS = "25"        %{ maximum number of rows in the table  %}
   resultTable = %table       %{ table to hold query results          %}
%}
```

DEFINE Block or Statement

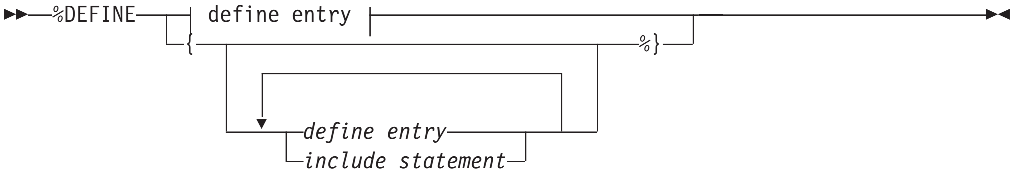
Purpose

The DEFINE section defines variables names in the declaration part of the macro and can be either a statement or a block.

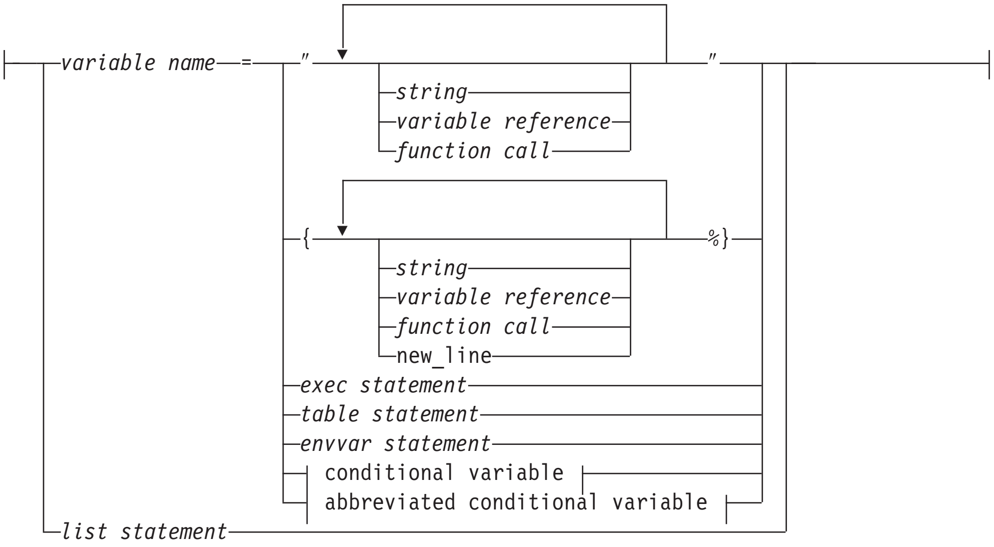
- Use statements to define one variable at a time
- Use blocks to define several variables

The variable definition can be on a single line, using double quotes (″″), or can span multiple lines, using brackets and a percent sign ({ %}). After the variable is defined, you can reference it anywhere in the macro.

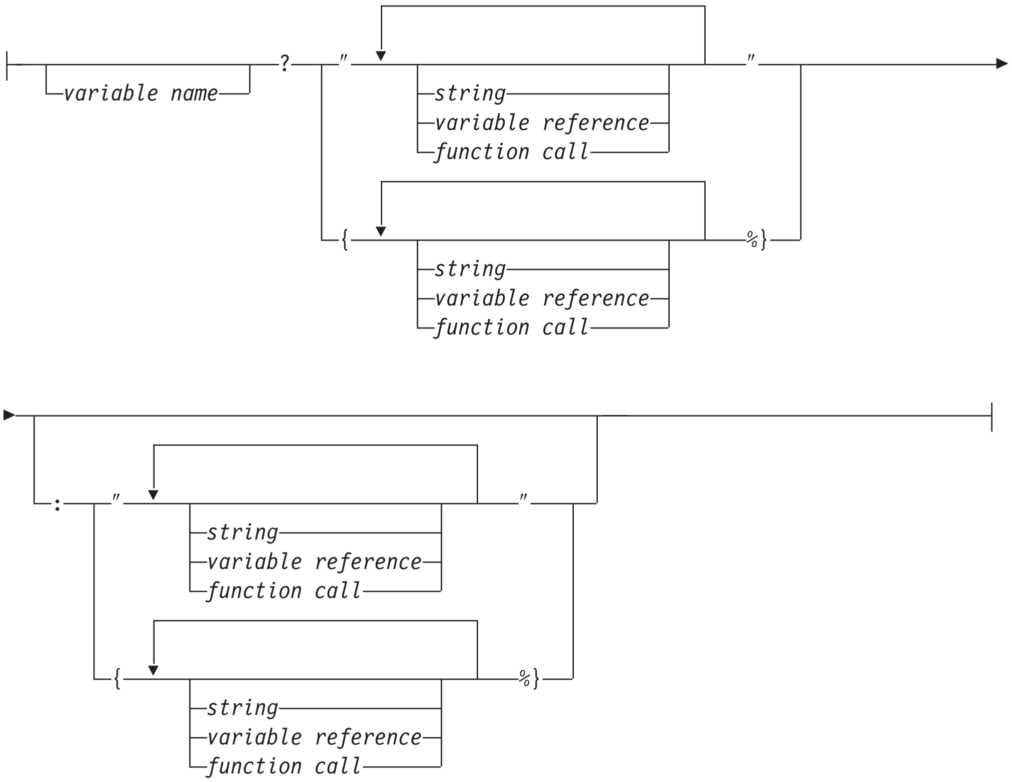
Syntax



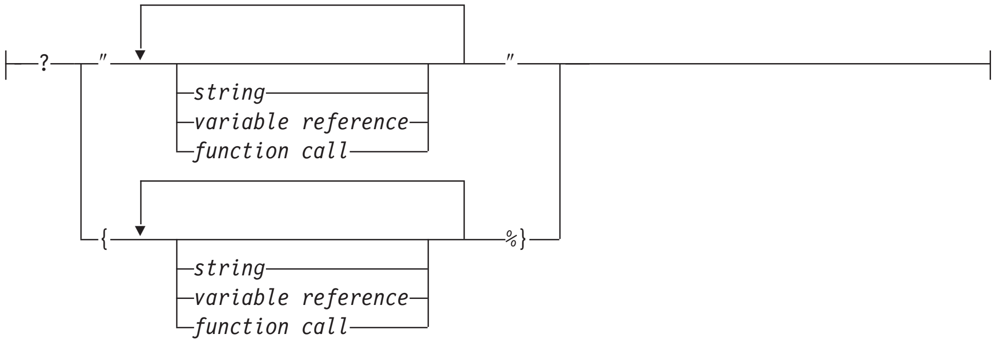
define entry



conditional variable



abbreviated conditional variable



Values

%DEFINE
A keyword that defines variables.

define entry:

variable name
One or more names, each additional name concatenated by a period (.). See “Variable Name” on page 4 for syntax information.

string
Any sequence of alphabetic and numeric characters and punctuation. If the string appears within double quotes, the new-line character is not allowed.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

| **function call**
| Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function Call (@)” on page 22 for syntax and examples.

exec statement
The EXEC statement. The name of an external program that executes when a variable is referenced or a function is called. See “EXEC Block or Statement” on page 14 for syntax and examples.

table statement
The TABLE statement. Defines a collection of related data containing an array of identical records, or rows, and an array of column names describing the fields in each row. See “TABLE Statement” on page 52 for syntax and examples.

envvar statement
The ENVVAR statement. Refers to environment variables. See “ENVVAR Statement” on page 13 for syntax and examples.

conditional variable
Sets the value of a variable based on the value of another variable or string.

| **abbreviated conditional variable**
| Sets the value of a variable based on the value of another variable or string. A shorter form of the conditional variable.

list statement
The LIST statement. Defines variables that are used to build a delimited list of values. See “LIST Statement” on page 36 for syntax and examples.

include statement
The INCLUDE statement. Reads and incorporates a file into the Net.Data macro. See “INCLUDE Statement” on page 32 for syntax and examples.

Context

The DEFINE block or statement must be in an IF block or outside all other blocks in the declaration part of the Net.Data macro.

Restrictions

- Can contain the following elements:
  - | Comment block
  - Conditional variables
  - LIST statement
  - TABLE statement
  - Variable references
  - INCLUDE statement
  - EXEC statement
  - Function calls
  - ENVVAR statement

- The conditional variable cannot have a result of NULL. See Example 5 for more information.
- You cannot use a variable in its own definition. For example, the following variable definition is not allowed:

  ```
  %DEFINE var = "The value is $(var)."
  ```

Examples

Example 1: Simple variable definitions

```
%DEFINE var1 = "orders"
%DEFINE var2 = "$(var1).html"
```

During run time, the variable reference *$(var2)* is evaluated as *orders.html*.

Example 2: Quotes inside a string

```
%DEFINE hi = "say ""hello"""
%DEFINE empty = ""
```

When displayed, the variable *hi* has the value *say "hello"*. The variable *empty* is null.

Example 3: Definition of multiple variables

```
%DEFINE{  DATABASE = "testdb"
          home = "http://www.software.ibm.com"
          SHOWSQL = "YES"
          PI = "3.14150"
%}
```

Example 4: Multiple-line definition of a variable

```
%DEFINE text = {This variable definition
            spans two lines
%}
```

Example 5: This example of a conditional variable demonstrates how the variable `var` takes the resulting value inside the quotations marks (“”) if the resulting value does not contain any NULL values. In the example below, neither $(V) nor MyFunc can have a result of NULL.

```
%DEFINE var = ? "Hello! $(V)@MyFunc()"
%}
```

ENVVAR Statement

Purpose

Defines a variable as an environment variable in the DEFINE block. When the ENVVAR variable is referenced, Net.Data returns the current value of the environment variable by the same name. Using this method to reference environment variables is more efficient than using DTW_GETENV. For more information about DTW_GETENV, see “DTW_GETENV” on page 123.

Syntax

```
►►──%ENVVAR──────────────────────────────────────────────────────────────►◄
```

Context

The ENVVAR statement can be in the DEFINE block or statement.

Values

%ENVVAR
The keyword for defining a variable as an environment variable in a DEFINE block. This variable gets the value of an environment variable anywhere in the macro file.

Restrictions

The ENVVAR statement can contain no other elements.

Examples

Example 1: In this example, ENVVAR defines a variable, which when referenced, returns the current value for the environment variable SERVER_SOFTWARE, the name of the Web server.

```
%DEFINE SERVER_SOFTWARE = %ENVVAR

%HTML (REPORT){
The server is $(SERVER_SOFTWARE).
%}
```

EXEC Block or Statement

Purpose

Specifies an external program to execute when a variable is referenced or a function is called.

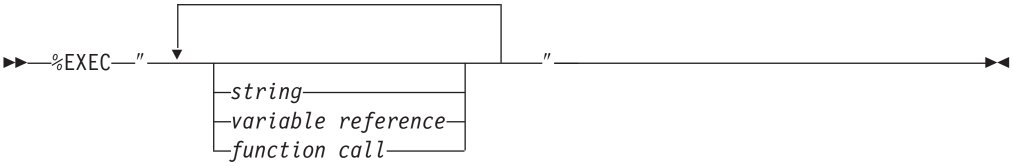
When a variable is referenced or a function called, Net.Data first looks up the directories specified in the EXEC_PATH variable in the Net.Data initialization file and, when not found there, passes the name of the executable to the system shell.

Authorization Tip: Ensure that the Web server has access rights to any files referenced by the EXEC statement or block. See the section on specifying Web server access rights to Net.Data files in the configuration chapter of *Net.Data Administration and Programming Guide* for more information.

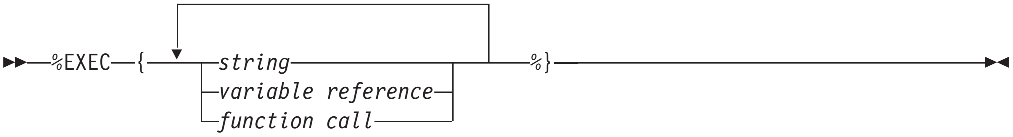
The EXEC statement and block are used in two different contexts and have different syntax, depending where they are used. Use the EXEC statement in the DEFINE block, and use the EXEC block in the FUNCTION block.

Syntax

The EXEC statement syntax when used in the DEFINE block:



The EXEC block syntax when used in the FUNCTION block:



Values

%EXEC
The keyword that specifies the name of an external program to be executed when a variable is referenced or when a function is called. When Net.Data encounters a variable reference that is defined in an EXEC statement, it processes what the EXEC statement declares for the variable.

string
Any sequence of alphabetic and numeric characters and punctuation. If the string appears within double quotes, the new-line character is not allowed.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function Call (@)” on page 22 for syntax and examples.

Context

The EXEC block or statement can be found in these contexts:

- DEFINE block
- FUNCTION block

Restrictions

The EXEC block or statement can contain these elements:

- Comment block
- String
- Variable references
- Function call

Examples

Example 1: Executable file referenced by a variable

```
%DEFINE mycall = %EXEC "MYEXEC.EXE $(empno)"

%HTML (report){
<P>Here is the report you requested:
<HR>$(mycall)
%}
```

This example executes `MYEXEC.EXE` on every reference to the variable, mycall.

Example 2: Executable file referenced by a function

```
%FUNCTION(DTW_REXX) my_rexx_pgm(INOUT a, b, IN c, INOUT d){
   %EXEC{ mypgm.cmd this is a test %}
%}
```

This example executes `mypgm.cmd` when the function `my_rexx_pgm` is called.

FUNCTION Block

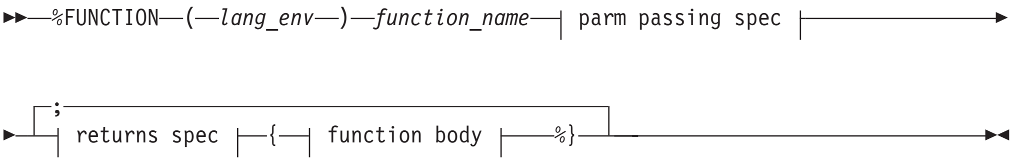
Purpose

Defines a subroutine that Net.Data invokes from the macro file. The executable statements in a FUNCTION block can be inline statements directly interpreted by a language environment, or they can be a call to an external program.

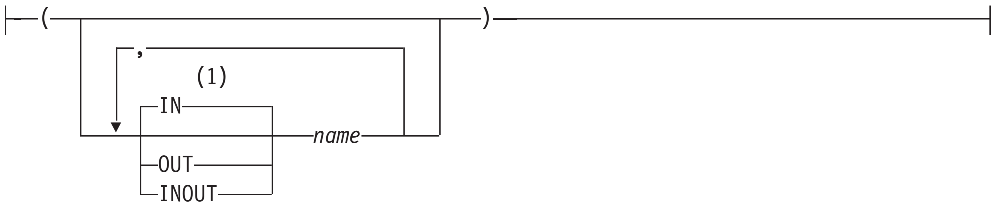
If you use the EXEC block within the FUNCTION block, it must be the only executable statement in the FUNCTION block. Before passing the executable statement to the language environment, Net.Data appends the file name of the program in the EXEC block to a path name determined by the EXEC_PATH configuration statement in the initialization file. The resulting string is passed to the language environment to be executed.

The method that the language environment uses to process the EXEC block depends on the particular language environment. Only the REXX, System, and Perl Net.Data-provided language environments support the EXEC block.

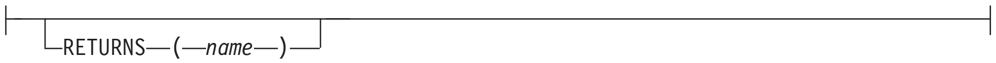
Syntax



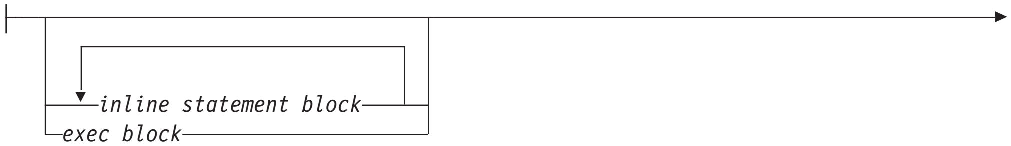
parm passing spec

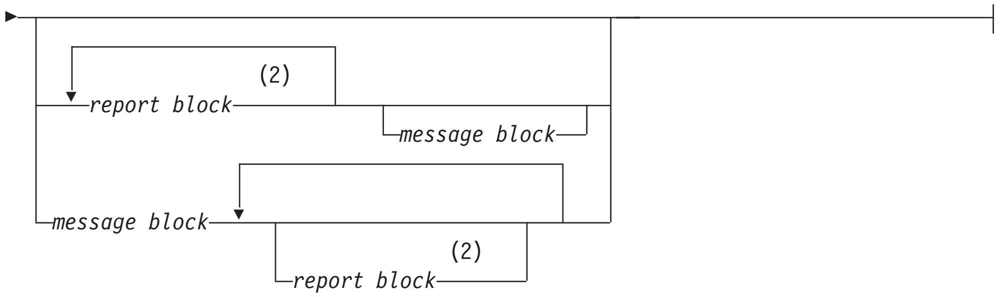


returns spec



function body





Notes:

1. The default parameter type of IN applies when no parameter type is specified at the beginning of the parameter list. A parameter without a parameter type uses the type most recently specified in the parameter list, or type IN if no type has been specified. For example, in the parameter list (`*parm1*, INOUT *parm2*, *parm3*, OUT *parm4*, *parm5*`), parameters *parm1*, *parm3*, and *parm5* do not have parameter types. The parameter *parm1* has a type of IN because no initial parameter type has been specified. The parameter *parm3* has a type of INOUT because it is the most recently specified parameter type. Similarly, the parameter *parm5* has a type of OUT because it is the most recently specified type in the parameter list.
2. The repeated report block is only valid for database language environments when processing stored procedures that return multiple result sets.

Values

%FUNCTION
The keyword that specifies a subroutine that Net.Data invokes from the macro file.

lang_env
The language environment that processes the function body. See the *Net.Data Language Environment Reference* for more information.

function_name
The name of the function being defined that can be an alphabetic or numeric string that begins with an alphabetic character or underscore and contains any combination of alphabetic, numeric, or underscore characters.

name
An alphabetic or numeric string beginning with an alphabetic character or underscore and containing any combination of alphabetic, numeric, or underscore characters.

parm passing spec:

IN Specifies that Net.Data passes input data to the language environment. IN is the default.

OUT
Specifies that the language environment returns output data to Net.Data.

INOUT
Specifies that Net.Data passes input data to the language environment and the language environment returns output data to Net.Data.

returns spec:

RETURNS

Declares the variable that contains the function value assigned by the language environment, after the function completes.

function body:

inline statement block

Syntactically valid statements from the language environment specified in the function definition, for example; REXX, SQL, or Perl. See *Net.Data Language Environment Reference* for a description of the language environment you are using. See the programming language's programming reference for syntax and usage. The string representing the inline statement block can contain Net.Data variable references and function calls, which get evaluated before execution of the inline statement block (program). **Restriction:** The longest consecutive inline statement block string without any Net.Data variable reference or function call is limited to the following lengths:

- For OS/2 and NT: 64KB
- For AIX: 256KB
- For OS/390: 256KB
- For OS/400: 256KB

exec block

The EXEC block. The name of an external program that executes when a variable is referenced or a function is called. See "EXEC Block or Statement" on page 14 for syntax and examples.

report block

The REPORT block. Formatting instructions for the output of a function call. You can use header and footer information for the report. See "REPORT Block" on page 47 for syntax and examples.

message block

The MESSAGE block. A set of return codes, the associated messages, and the actions Net.Data takes when a function call is returned. See "MESSAGE Block" on page 42 for syntax and examples.

Context

The FUNCTION block can be found in these contexts:

- IF block
- Outside of any block or statement in the declaration part of the Net.Data macro.

Restrictions

The FUNCTION block can contain these elements:

- Comment block
- EXEC block
- MESSAGE block
- REPORT block
- Inline statement blocks

Only the REXX, System, and Perl Net.Data-provided language environments support the EXEC statement.

Examples

The following examples are general and do not cover all language environments. See *Net.Data Language Environment Reference* for more information about using FUNCTION blocks with a specific language environment.

Example 1: A REXX substring function

```
%DEFINE lstring = "longstring"
%FUNCTION(DTW_REXX) substring(IN x, y, z) RETURNS(s) {
  s = substr("$(x)", $(y), $(z));
%}
%DEFINE a = {@substring(lstring, "1", "4")%} %{ assigns "long" to a %}
```

When *a* is evaluated, the @substring function call is found and the substring FUNCTION block is executed. Variables are substituted in the executable statements in the FUNCTION block, then the text string `s = substr("longstring", 1, 4)` is passed to the REXX interpreter to execute. Because the RETURNS clause is specified, the value of the @substring function call in the evaluation of *a* is replaced with “long”, the value of *s*.

Example 2: Invoking an external REXX program

- Net.Data macro:

  ```
  %FUNCTION(DTW_REXX) my_rexx_pgm(INOUT a, b, IN c, OUT d) {
   %EXEC{ mypgm.cmd this is a test %}
  %}
  %HTML(INPUT) {
   <P> Original variable values: $(w) $(x) $(z)
   <P> @my_rexx_pgm(w, x, y, z)
   <P> Modified variable values: $(w) $(x) $(z)
  %}
  ```

  Variables *w* and *x* correspond to the INOUT parameters *a* and *b* in the function. Their values and the value of *y*, which corresponds to the IN parameter *c*, should already be defined from HTML form input or from a DEFINE statement. Variables *a* and *b* are assigned new values when parameters *a* and *b* return values. The variable *z* is defined when the OUT parameter *d* returns a value.

- REXX program `mypgm.cmd`:

  ```
  /* Sample REXX Program for Example 2 */
  /* Test arguments */
  num_args = arg();
  say ’There are’ num_args ’arguments’;
  do i = 1 to num_args;
    say ’arg’ i ’is "’arg(i)’"’
  end;
  /* Set variables passed from Net.Data */
  d = a || b || c;     /* concatenate a, b, and c forming d */
  a = ’’;             /* reset a to null string */
  b = ’’;             /* reset b to null string */
  return;
  ```

- Output from `mypgm.cmd`:

  ```
  There are 1 arguments
  arg 1 is "this is a test"
  ```

The EXEC statement tells the REXX language environment to tell the REXX interpreter to execute the external REXX program `mypgm.cmd`. Because the REXX language environment can directly share Net.Data variables with the REXX program, it assigns the REXX variables *a*, *b*, and *c* the values of the Net.Data variables *w*, *x* and *y* before executing `mypgm.cmd`. `mypgm.cmd` can directly use the variables *a*, *b*, and *c* in REXX statements. When the program ends, the REXX variables *a*, *b*, and *d* are retrieved from the REXX program, and their values are assigned to the Net.Data variables *w*, *x*, and *z*. Because the RETURNS clause is not used in the definition of the `my_rexx_pgm` FUNCTION block, the value of the `@my_rexx_pgm` function call is the null string, “”, (if the return code is 0) or the value of the REXX program return code (if the return code is nonzero).

Example 3: An SQL query and report

```
%FUNCTION(DTW_SQL) query_1(IN x, IN y) {
   SELECT customer.num, order.num, part.num, status
   FROM customer, order, shippingpart
   WHERE customer.num = ’$(x)’
      AND customer.ordernumber = order.num
      AND order.num = ’$(y)’
      AND order.partnumber = part.num
   %REPORT{
     <P>Here is the status of your order:
     <P>$(NLIST)
     <UL>
     %ROW{
       <LI>$(V1)  $(V2)  $(V3)  $(V4)
     %}
     </UL>
        %}
%}
%DEFINE customer_name="IBM"
%DEFINE customer_order="12345"
%HTML(REPORT) {
  @query_1(customer_name, customer_order)
%}
```

The `@query_1` function call substitutes `IBM` for *$(x)* and `12345` for *$(y)* in the SELECT statement. Because the definition of the SQL function `query_1` does not identify an output table variable, the default table is used (see the TABLE variables block for details). The NLIST and Vi variables referenced in the REPORT block are defined by the default table definition. The report produced by the REPORT block is placed in the output HTML where the `query_1` function is invoked.

Example 4: A system call to execute a Perl script

- Net.Data macro:

  ```
  %FUNCTION(DTW_SYSTEM) today() RETURNS(result) {
    %exec{ perl "today.prl" %}
  %}
  %HTML(INPUT) {
    @today()
  %}
  ```

- Perl program `today.prl`:

  ```
  $date = ′date′;
    chop $date;
    open(DTW, "> $ENV{DTWPIPE}") || die "Could not open: $!";
    print DTW "result = \"$date\"\n";
  ```

The System language environment interprets the executable statements in a FUNCTION block by passing them to the operating system through the C language system() function call. This method does not allow Net.Data variables to be directly passed or retrieved to the executable statements, as the REXX language environment does, so the System language environment passes and retrieves variables as described here:

- Input parameters are passed as system environment variables through the putenv() function and can be retrieved by the executing program. Different languages reference the variables differently. A UNIX cshell script refers to environment variables by preceding the environment variable name with a ’$’, such as `$x`. A Perl language script refers to them by referencing the associative array %ENV, such as `%ENV{’x’}`. A DOS batch (.BAT) file refers to the variable name enclosed in percent signs, such as `%x%`.

- Output parameters are passed back to the language environment by writing to a pipe whose name is passed in the environment variable DTWPIPE, except on the OS/400 platform, where output parameters are passed back to the language environment as system environment variables. The data that is written to the named pipe has the form `name="value"`, just as with DEFINE statements. If a variable name corresponding to an output parameter is written this way, the new value replaces the current value. If a variable name is written that does not correspond to an output parameter, it is ignored.

When the `@today` function call is encountered, Net.Data performs variable substitution on the executable statements. In this example, there are no Net.Data variables in the executable statements, so no variable substitution is performed. The executable statements and parameters are passed to the System language environment, which creates a named pipe and sets the environment variable DTWPIPE to the name of the pipe.

Then the external program is called with the C system() function call. The external program opens the pipe as write-only and writes the values of output parameters to the pipe as if it were a standard stream file. The external program generates HTML output by writing to STDOUT. In this example, the output of the system date program is assigned to the variable result, which is the variable identified in the RETURNS clause of the FUNCTION block. This value of the result variable replaces the `@today()` function call in the HTML block.

Example 5: Perl language environment

```
%FUNCTION(DTW_PERL) today() RETURNS(result) {
  $date = ′date′;
  chop $date;
  open(DTW, "> $ENV{DTWPIPE}") || die "Could not open: $!";
  print DTW "result = \"$date\"\n";
%}
%HTML(INPUT) {
  @today()
%}
```

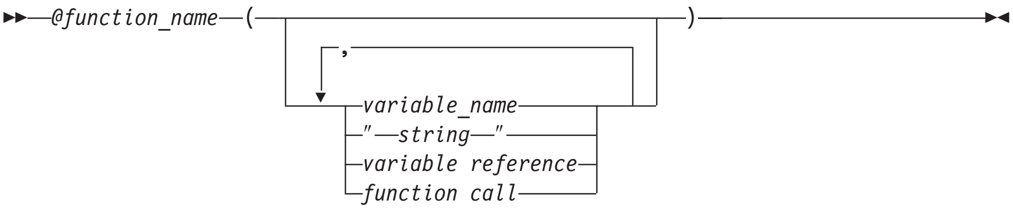
Compare this example with Example 4 to see how the EXEC block is used. In Example 4, the System language environment does not understand how to interpret Perl programs, but the language environment does know how to call external programs. The EXEC block tells it to call a program called `perl` as an external program. The actual Perl language statements are interpreted by the external Perl program. Example 5 has no EXEC block, because the Perl language environment is able to directly interpret Perl language statements.

Function Call (@)

Purpose

Invokes a previously defined FUNCTION block, MACRO_FUNCTION block, or built-in function with specified arguments. If the function is not a built-in function, you must define it in the Net.Data macro before you specify a function call.

Syntax



Values

@function_name
The name of any existing function. An alphabetic or numeric string that begins with an alphabetic character or underscore and contains any combination of alphabetic, numeric, or underscore characters.

variable name
One or more names, each additional name concatenated by a period (.). See “Variable Name” on page 4 for syntax information.

string
Any sequence of alphabetic and numeric characters and punctuation, except the new-line character.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments.

Context

Function calls can be found in these contexts:

- HTML block
- REPORT block
- ROW block
- DEFINE block
- IF block
- MACRO_FUNCTION block
- MESSAGE block
- WHILE block
- Function call statement
- Outside of any block in the declaration part of the Net.Data macro

Restrictions

- Function calls can contain these elements:
  - Comment block
  - Strings
  - Function calls
  - Variable References
- Function calls cannot contain any variable references and function calls defined for OUT or INOUT parameters in a function definition.

Examples

Example 1: A call to the SQL function formQuery

```
%FUNCTION(DTW_SQL) formQuery(){
SELECT $(queryVal) from $(tableName)
%}

%HTML (input){
<P>Which columns of $(tableName) do you want to see?
<FORM METHOD="POST" ACTION="report">
<INPUT NAME="queryVal" TYPE="CHECKBOX" VALUE="NAME">Name
<INPUT NAME="queryVal" TYPE="CHECKBOX" VALUE="MAIL">E-mail
<INPUT NAME="queryVal" TYPE="CHECKBOX" VALUE="FAX">FAX
<INPUT TYPE="SUBMIT" VALUE="Submit request">
%}

%HTML (report){
<P>Here are the columns you selected:
<HR>@formQuery()
%}
```

Example 2: A call to a REXX function with input and output parameters

```
%FUNCTION(DTW_REXX) my_rexx_pgm(INOUT a, b, IN c, OUT d) {
 %EXEC{ mypgm.cmd this is a test %}
%}
%HTML(INPUT) {
 <P> Original variable values: $(w) $(x) $(z)
 <P> @my_rexx_pgm(w, x, y, z)
 <P> Modified variable values: $(w) $(x) $(z)
%}
```

Example 3: A call to a REXX function, with input parameters, that uses variable references and function calls

```
%FUNCTION(DTW_REXX) my_rexx_pgm(IN a, b, c, d, OUT e) {
  ...
%}
%HTML(INPUT) {
 <p>  @my_rexx_pgm($(myA), @getB(), @retrieveC(), $(myD), myE)
%}
```

HTML Block

Purpose

Contains any HTML tags or text to be processed by the client's Web browser or any tool that understands HTML. The HTML block can also contain most Net.Data macro language statements, which are evaluated and executed at run time. Net.Data looks for Net.Data macro statements and executes them. Net.Data assumes all other text is HTML and sends it to the Web browser.

Syntax

```
                            ┌──────────────────────────────┐
                            ▼                              │
►►──%HTML──(──name──)──{─────┬─────────────────────────┬───┴──%}──────────────────►◄
                             ├─exec_sql statement──────┤
                             ├─variable reference──────┤
                             ├─if block────────────────┤
                             ├─function call───────────┤
                             ├─HTML statement──────────┤
                             ├─include statement───────┤
                             ├─include_url statement───┤
                             └─while block─────────────┘
```

Values

%HTML
The keyword that specifies the block that contains HTML tags and text to be displayed on the client's browser.

name
An alphabetic or numeric string that begins with an alphabetic character or underscore and contains any combination of alphabetic, numeric, or underscore characters.

exec_sql statement
A DB2WWW Release 1 language element that is supported for compatibility. See "Appendix A. DB2 WWW Connection" on page 223 or DB2 World Wide Web Release 1 documentation.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See "Variable Reference" on page 4 for syntax information.

if block
The IF block. Performs conditional string processing. String values in the condition list are treated as numeric for comparisons if they are strings that represent integers and have no leading or trailing white space. They can have a single leading plus (+) or minus (-) sign. See "IF Block" on page 26 for syntax and examples.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See "Function Call (@)" on page 22 for syntax and examples.

HTML statements
Includes any alphabetic or numeric characters, as well as HTML tags to be formatted for the client's browser.

include statement
The INCLUDE statement. Reads and incorporates a file into the Net.Data macro. See "INCLUDE Statement" on page 32 for syntax and examples.

include_url statement
The INCLUDE_URL statement. Reads and incorporates another file into the Net.Data Web macro where the statement is specified. The specified file can exist on a local or remote server. See "INCLUDE_URL Statement" on page 34 for syntax and examples.

while block
The WHILE block. Performs looping with conditional string processing. See "WHILE Block" on page 54 for syntax and examples.

Context

The HTML block can be found in these contexts:

- IF block
- Outside of any block in the declaration part of the Net.Data macro

Restrictions

The HTML block can contain these elements:

- Comment block
- EXEC_SQL statement
- IF block
- HTML statements
- INCLUDE statement
- INCLUDE_URL statement
- WHILE block
- Variable references
- Function calls

Examples

Example 1: HTML block with include files for headings and footings

```
%HTML(example1){
%INCLUDE"header.html"
<P>You can put <EM>any</EM> HTML in an HTML block.
An SQL function call is made like this:
@xmp1()
%INCLUDE"footer.html"
%}
```

IF Block

Purpose

Performs conditional string processing. The IF block provides the ability to test one or more conditions, and then to perform a block of statements based on the outcome of the condition test. You can use the IF block in the declaration part of a Net.Data macro, the HTML block, the MACRO_FUNCTION block, the REPORT block, the WHILE block, and the ROW block, as well as nest it inside another IF block.

String values in the condition list are treated as numeric for comparisons if they are strings that represent integers and have no leading or trailing white space. They can have a single leading plus (+) or minus (-) sign.

Restriction: Net.Data does not support numerical comparison of non-integer numbers. For example, floating point numbers.

Nested IF blocks: The rules for IF block syntax are determined by the block's position in the macro file. If an IF block is nested within an IF block that is outside of any other block in the declaration part, it can use any element that the outside block can use. If an IF block is nested within another block that is in an IF block, it takes on the syntax rules for the block it is inside.

In the following example, the nested IF block must follow the rules used when it is inside an HTML block.

```
%IF block
...
  %HTML block
...
     %IF block
```

See the restrictions listed later in this section.

Syntax

```
►►──%IF──┤ condition list ├──┤ statement_block ├──┤ else_if spec ├──%ENDIF────────►◄
```

condition list

```
├──(──┬─(──condition list──)──────────────┬──)──────────────────────────────────┤
      ├─condition list──&&──condition list─┤
      ├─condition list──||──condition list─┤
      ├─!──condition list──────────────────┤
      ├─┤ condition ├──────────────────────┤
      └─┤ term ├───────────────────────────┘
```

statement_block

```
   ┌─────────────────────────────────┐
   ▼                                 │
├──────┬────────────────────────────┬──┴─────────────────────────────────┤
       │                (1)         │
       ├─define block───────────────┤
       │                    (1)     │
       ├─define statement───────────┤
       │                      (2)   │
       ├─exec_sql statement─────────┤
       │                  (1)       │
       ├─function block─────────────┤
       ├─function call──────────────┤
       │              (1)           │
       ├─HTML block─────────────────┤
       │                  (2)       │
       ├─HTML statement─────────────┤
       ├─if block───────────────────┤
       ├─include statement──────────┤
       ├─include_url statement──────┤
       │                        (1) │
       ├─macro_function block───────┤
       │                 (1)        │
       ├─message block──────────────┤
       │          (2)               │
       ├─string─────────────────────┤
       │                      (2)   │
       ├─variable reference─────────┤
       │               (2)          │
       └─while block────────────────┘
```

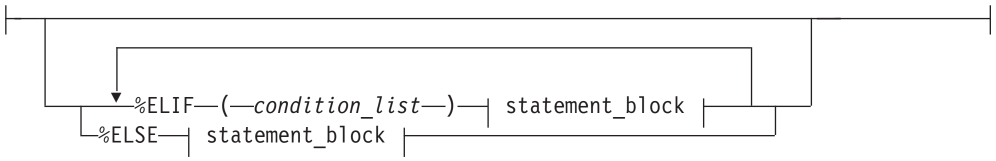
condition

```
├──term───┬─<──┬───term──────────────────────────────────────────────┤
          ├─>──┤
          ├─<=─┤
          ├─>=─┤
          ├─!=─┤
          └─==─┘
```

term

```
├───┬─variable reference─┬──────────────────────────────────────────┤
    ├─"──string──"───────┤
    ├─variable name──────┤
    └─function call──────┘
```

else_if spec



Notes:

1. This language construct is valid when the IF block is located outside of any other block in the declaration part of the macro.
2. This language construct is valid when the IF block is located in an HTML block, MACRO_FUNCTION block, REPORT block, or WHILE block.

Values

%IF
The keyword that specifies conditional string processing.

condition list
Compares the values of conditions and terms. Condition lists can be connected using Boolean operators. A condition list can be nested inside another condition list.

statement_block
The following valid Net.Data macro constructs. Please see diagram notes and restrictions to determine the context in which the macro constructs are valid.

define statement
The DEFINE block or statement. Defines variables and sets configuration variables. Variable names must begin with a letter or underscore (_) and contain any alphanumeric characters or underscore. See “DEFINE Block or Statement” on page 9 for syntax and examples.

exec_sql statement
A DB2WWW Release 1 language element that is supported for compatibility. See “Appendix A. DB2 WWW Connection” on page 223 or DB2 World Wide Web Release 1 documentation.

function block
A keyword that specifies a subroutine that can be invoked from the Net.Data macro. The executable statements in a FUNCTION block can contain language statements that are directly interpreted by a language environment, or they can indicate a call to an external program. See “FUNCTION Block” on page 16 for syntax and examples.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function Call (@)” on page 22 for syntax and examples.

HTML block
Includes any alphabetic or numeric characters, as well as HTML tags to be formatted for the client’s browser.

HTML statement
Includes any alphabetic or numeric characters, and HTML tags to be formatted for the client’s browser.

if block

The IF block. Performs conditional string processing. String values in the condition list are treated as numeric for comparisons if they are strings that represent integers and have no leading or trailing white space. They can have a single leading plus (+) or minus (-) sign.

include statement

The INCLUDE statement. Reads and incorporates a file into the Net.Data macro. See “INCLUDE Statement” on page 32 for syntax and examples.

include_url statement

The INCLUDE_URL statement. Reads and incorporates another file into the Net.Data Web macro where the statement is specified. The specified file can exist on a local or remote server. See “INCLUDE_URL Statement” on page 34 for syntax and examples.

macro_function block

A keyword that specifies a subroutine that can be invoked from the Net.Data macro. The executable statements in a MACRO_FUNCTION block can contain Net.Data macro language source statements. See “MACRO_FUNCTION Block” on page 38 for syntax and examples.

message block

The MESSAGE block. A set of return codes, the associated messages, and the actions Net.Data takes when a function call is returned. See “MESSAGE Block” on page 42 for syntax and examples.

string

Any sequence of alphabetic and numeric characters and punctuation. If the string is in the term of the condition list, it can contain any character except the new-line character. If the string is in the executable block of code, it can contain any character, including the new-line character.

variable reference

Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

while block

The WHILE block. Performs looping with conditional string processing. See “WHILE Block” on page 54 for syntax and examples

condition

A comparison between two terms using comparison operators. An IF condition is treated as a numeric comparison if both of the following conditions are true:

- The condition operator is one of the following operators: <,<=,>,>=,==,!=
- Both terms are strings representing valid integers, where a valid integer is a string of digits, optionally preceded by a plus (+) or minus (-) sign, and no other white space.

If either condition is not true, a normal string comparison is performed.

term

A variable name, string, variable reference, or function call.

%ELIF

A keyword that starts the alternative processing path and can contain condition lists and most Net.Data macro statements.

%ENDIF

A keyword that closes the %IF block.

%ELSE
A keyword that executes associated statements if all other condition lists are not satisfied.

Context

The IF block can be found in these contexts:
- Outside of any other block in the declaration part of a Net.Data macro
- HTML block
- IF block
- MACRO_FUNCTION block
- REPORT block
- ROW block
- WHILE block

Restrictions

The IF block can contain these elements when located outside of any other block in the declaration part of the Net.Data macro:
- Comment block
- DEFINE block
- DEFINE statement
- FUNCTION block
- Function call
- HTML block
- IF block
- INCLUDE statement
- INCLUDE_URL statement
- MACRO_FUNCTION block
- MESSAGE block
- Variable reference

The IF block can contain these elements when located in the HTML block, MACRO_FUNCTION block, REPORT block, ROW block, or WHILE block of the Net.Data macro:
- Comment block
- EXEC_SQL statement
- Function calls
- IF block
- INCLUDE statement
- INCLUDE_URL statement
- HTML statement
- String
- Variable reference
- WHILE block

Examples

Example 1: An IF block in the declaration part of a Net.Data macro

```
%DEFINE a = "1"
%DEFINE b = "2"
...
%IF ($(DTW_HTML_TABLE) == "YES")
  %define OUT_FORMAT = "HTML"
%ELSE
  %define OUT_FORMAT = "CHARACTER"
%ENDIF

%HTML(REPORT) {
 ...
%}
```

Example 2: An IF block inside an HTML block

```
%HTML(REPORT) {
@myFunctionCall()
%IF ($RETURN_CODE) == $(failure_rc))
   <P> The function call failed with failure code $(RETURN_CODE).
%ELIF ($(RETURN_CODE) == $(warning_rc))
   <P> The function call succeeded with warning code $(RETURN_CODE).
%ELIF ($(RETURN_CODE) == $(success_rc))
   <P>The function call was successful.
%ELSE
   P>The function call returned with unknown return code $(RETURN_CODE).
%ENDIF
%}
```

Example 3: A numeric comparison

```
%IF (ROW_NUM < "100")
    <p>The table is not full yet...
%ELIF (ROW_NUM == "100")
    <p>The table is now full...
%ELSE
    <p>The table has overflowed...
%ENDIF
```

A numeric comparison is done because the implicit table variable ROW_NUM always returns an integer value, and the value that is being compared is also an integer.

Example 4: Nested IF blocks

```
%IF (MONTH == "January")
  %IF (DATE = "1")
    HAPPY NEW YEAR!
  %ELSE
    Ho hum, just another day.
  %ENDIF
%ENDIF
```

INCLUDE Statement

Purpose

Reads and incorporates a file into the Net.Data macro in which the statement is specified.

Net.Data searches the directories specified in the INCLUDE_PATH statement in the initialization file to find the include file.

You can use include files the same way you can in most high-level languages. They can insert common headings and footings, define common sets of variables, or incorporate a common subroutine library of FUNCTION block definitions into a Net.Data macro.

Net.Data executes an INCLUDE statement only once when processing the macro and inserts the content of the included file at the location of the INCLUDE statement in the macro file. Any variable references in the name of the included file are resolved at the time the INCLUDE statement is first executed, not when the content of the included file is to be executed.

When an INCLUDE statement is in a ROW or WHILE block, Net.Data does not repeatedly execute the INCLUDE statement. Net.Data executes the INCLUDE statement the first time it executes the ROW or WHILE block, incorporates the content of the included file into the block, and then repeatedly executes the ROW or WHILE block with the content of the included file.

Authorization Tip: Ensure that the Web server has access rights to any files referenced by the INCLUDE statement. See the section on specifying Web server access rights to Net.Data files in the configuration chapter of *Net.Data Administration and Programming Guide* for more information.

Tip: If you want to include an HTML file from a local Web server, use the INCLUDE_URL construct as shown in Example 3 for INCLUDE_URL. By using the demonstrated syntax, you do not have to update the INCLUDE_PATH in the Net.Data initialization file to specify directories that are already known to the Web server.

Syntax

```
                   ┌──────────────────────────┐
                   ▼                          │
►►──%INCLUDE──"────┬─string─────────────┬─────┴──"──────────────────────────►◄
                   └─variable reference─┘
```

Values

%INCLUDE
The keyword that indicates a file is to be read and incorporated into the Net.Data macro.

name
An alphabetic or numeric string beginning with an alphabetic character or underscore and containing any combination of alphabetic, numeric, or underscore characters.

string
Any sequence of alphabetic and numeric characters and punctuation, except the new-line character.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

Context

The INCLUDE statement can be found in these contexts:

- DEFINE block
- HTML block
- REPORT block
- ROW block
- IF block
- MESSAGE block
- MACRO_FUNCTION block
- WHILE block
- Outside of any block in the declaration part of the Net.Data macro

Restrictions

The INCLUDE statement can contain these elements:

- Comment block
- Strings
- Variable references

Examples

Example 1: An INCLUDE statement in an HTML block

```
%HTML(start){
%INCLUDE "header.hti"
...
%}
```

Example 2: An INCLUDE statement in a REPORT block

```
%REPORT {
  %INCLUDE "report_header.txt"
  %ROW {
    %INCLUDE "row_include.txt"
  %}
  %INCLUDE "report_footer.txt"
%}
```

Example 3: Variable references in an INCLUDE statement

```
%define library = "/qsys.lib/mylib.lib/"
%define filename = "macros.file/incfile.mbr"

%include "$(library)$(filename)"
```

INCLUDE_URL Statement

Purpose

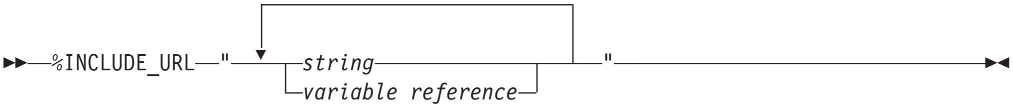
Reads and incorporates another file into the Net.Data generated output in which the statement is specified. The specified file can exist on a local or remote server.

Using the INCLUDE_URL statement, you can invoke one macro from another macro without requiring the application user to select a Submit button.

Net.Data executes an INCLUDE_URL statement only once when processing the macro and inserts the content of the included file at the location of the INCLUDE_URL statement in the macro file. Any variable references in the name of the included file are resolved at the time the INCLUDE_URL statement is first executed, not when the content of the included file is to be executed.

When an INCLUDE_URL statement is in a ROW or WHILE block, Net.Data does not repeatedly execute the INCLUDE_URL statement. Net.Data executes the INCLUDE_URL statement the first time it executes the ROW or WHILE block, incorporates the content of the included file into the block, and then repeatedly executes the ROW or WHILE block with the content of the included file.

Syntax



Values

%INCLUDE_URL
The keyword that indicates that a file is to be read and incorporated into the Net.Data macro from the local or a remote server.

string
Any sequence of alphabetic and numeric characters and punctuation, except the new-line character.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

Context

INCLUDE_URL statements can be found in these contexts:

- HTML block
- REPORT block
- ROW block
- WHILE block
- MACRO_FUNCTION block
- Outside any block in the declaration part of the Net.Data macro

Restrictions

INCLUDE_URL statements can contain these elements:

- Comment block
- Strings
- Variable references

The INCLUDE_URL file has the following file size limitations:

- OS/2 and Windows NT: 64 KB
- AIX: 256 KB
- OS/390: 256 KB

INCLUDE_URL is *not* supported in the OS/400 environment.

Examples

Example 1: Including an HTML file from another server

```
%include_url "http://www.ibm.com/path/myfile.html"
```

Example 2: Including an HTML file from a remote server by calling the server name

```
%include_url "myserver/path/myfile.html"
```

Where `myserver` is the server name.

Example 3: Including an HTML file from the local Web server

```
%include_url "/path/myfile.html"
```

Tip: By using this method, you do not have to update the INCLUDE_URL path in the Net.Data configuration file to specify directories that are already known to the Web server. If the *string* does *not* begin with a slash, Net.Data assumes the string is a server name and attempts to retrieve the file from the server with the corresponding name.

Example 4: Including other Net.Data macros from a remote server

```
%REPORT{
<P>Current hot pick as of @DTW_rTIME():
%include_url "http://www.ibm.com/cgi-bin/db2www/hotpic.mac/report?custno=$(custno)"
```

In this example, the macro file `hotpic.mac` is called and `custno` is sent as a variable. If the *string* begins with a slash, Net.Data retrieves the INCLUDE file from the local Web server.

LIST Statement

Purpose

Builds a delimited list of values. You can use the LIST statement when you construct SQL queries with multiple items like those found in some WHERE or HAVING clauses.

Syntax

```
                  ┌──────────────────────────┐
                  ▼                          │
►►──%LIST──"──────┬──────────────────────┬───┴──"────variable name──────────────────►◄
                  ├─string───────────────┤
                  ├─variable reference───┤
                  └─function call────────┘
```

Values

%LIST
The keyword that specifies that variables are to be used to build a delimited list of values.

string
Any sequence of alphabetic and numeric characters and punctuation, except the new-line character.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function Call (@)” on page 22 for syntax and examples.

variable name
One or more names, each additional name concatenated by a period (.). See “Variable Name” on page 4 for syntax information.

Context

The LIST statement can be found in these contexts:

- DEFINE statement

Restrictions

The LIST statement can contain these elements:

- Comment block
- Variable references
- Function calls
- Strings

Examples

Example 1: A list of variables

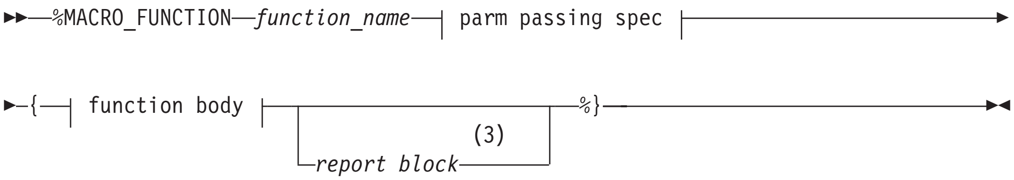
```
 %DEFINE{
DATABASE="custcity"
%LIST " OR " conditions
cond1="cond1='Sao Paolo'"
cond2="cond2='Seattle'"
cond3="cond3='Shanghai'"
whereClause=conditions ? "WHERE $(conditions)" : ""
%}
```

MACRO_FUNCTION Block

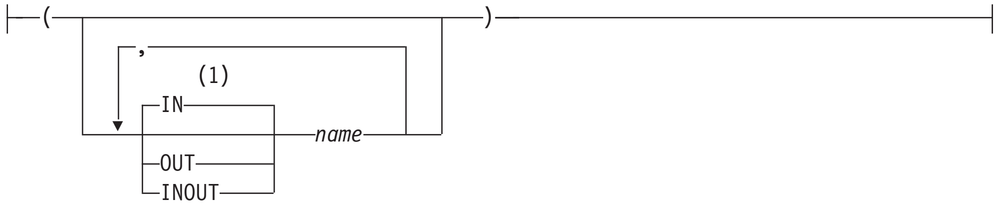
Purpose

Defines a subroutine that can be invoked from the Net.Data macro. The executable statements in a MACRO_FUNCTION block must be Net.Data macro language source statements.

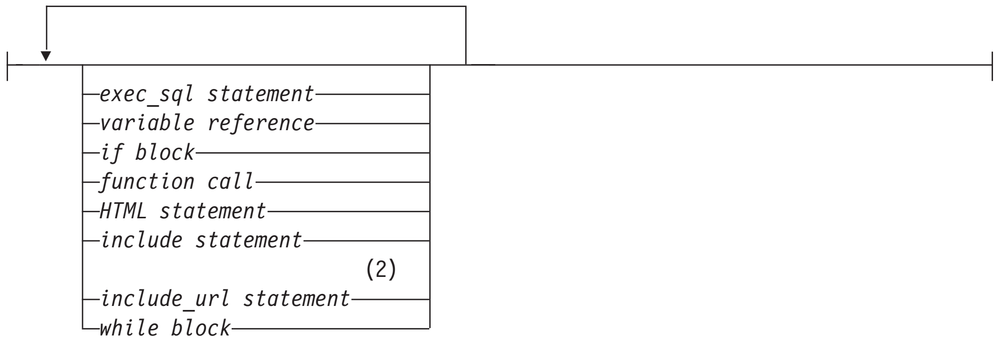
Syntax



parm passing spec



function body



Notes:

1. The default parameter type of IN applies when no parameter type is specified at the beginning of the parameter list. A parameter without a parameter type uses the type most recently specified in the parameter list, or type IN if no type has been specified. For example, in the parameter list (`parm1, INOUT parm2, parm3, OUT parm4, parm5`), parameters *parm1*, *parm3*, and *parm5* do not have parameter types. The parameter *parm1* has a type of IN because no initial parameter type has been specified. The parameter *parm3* has a type of INOUT because it is the most recently specified parameter type. Similarly, the parameter *parm5* has a type of OUT because it is the most recently specified type in the parameter list.

2. The INCLUDE_URL statement is not supported by OS/400.
3. The REPORT block is supported in the MACRO_FUNCTION block by OS/400 only.

Values

%MACRO_FUNCTION
The keyword that specifies a subroutine that can be invoked from the Net.Data macro. The executable statements in a MACRO_FUNCTION block must contain language statements that Net.Data directly interprets.

function_name
The name of the function being defined. An alphabetic or numeric string that begins with an alphabetic character or underscore and contains any combination of alphabetic, numeric, or underscore characters.

parm passing spec:

IN Specifies that Net.Data passes input data to the language environment. IN is the default.

OUT
Specifies that the language environment returns output data to Net.Data.

INOUT
Specifies that Net.Data passes input data to the language environment and the language environment returns output data to Net.Data.

name
An alphabetic or numeric string beginning with an alphabetic character or underscore and containing any combination of alphabetic, numeric, or underscore characters.

function body:

exec_sql
A DB2WWW Release 1 language element that is supported for compatibility. See “Appendix A. DB2 WWW Connection” on page 223 or DB2 World Wide Web Release 1 documentation.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

if block
The IF block. Performs conditional string processing. String values in the condition list are treated as numeric for comparisons if they represent integers and have no leading or trailing white space. They might have one leading plus (+) or minus (-) sign.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function Call (@)” on page 22 for syntax and examples.

HTML statement
Includes any alphabetic or numeric characters, as well as HTML tags to be formatted for the client’s browser.

include statement
The INCLUDE statement. Reads and incorporates a file into the Net.Data macro. See “INCLUDE Statement” on page 32 for syntax and examples.

include_url statement

The INCLUDE_URL statement. Reads and incorporates another file into the Net.Data macro in which the statement is specified. The specified file can exist on a local or remote server. See “INCLUDE_URL Statement” on page 34 for syntax and examples.

while block

The WHILE block. Performs looping with conditional string processing. See “WHILE Block” on page 54 for syntax and examples.

report block

The REPORT block. Formatting instructions for the output of a function call. You can use header and footer information for the report. See “REPORT Block” on page 47 for syntax and examples.

Context

The MACRO_FUNCTION block can be found in these contexts:

- IF block
- Outside of any block in the declaration part of the Net.Data macro

Restrictions

This construct is not available for the OS/390 operating system.

The MACRO_FUNCTION block can contain these elements:

- Comment block
- EXEC_SQL statement
- HTML statements
- IF block
- INCLUDE statement
- INCLUDE_URL statement

  Not supported for OS/400
- REPORT block

  Support for OS/400 only
- WHILE block
- Variable references
- Function calls

Examples

Example 1: A macro function that specifies message handling

```
%MACRO_FUNCTION setMessage(IN rc, OUT message) {
%IF (rc == "0")
  @dtw_assign(message, "Function call was successful.")
%ELIF (rc == "-1")
  @dtw_assign(message, "Function failed, out of memory.")
%ELIF (rc == "-2")
  @dtw_assign(message, "Function failed, invalid parameter.")
%ENDIF
%}
```

Example 2: A macro function that specifies header information

```
%MACRO_FUNCTION setup(IN browserType) {
%{ call this function at the top of each HTML block in the macro %}
%INCLUDE "header_info.html"
@dtw_rdate()
%IF (browserType == "IBM")
  @setupIBM()
%ELIF (browserType == "MS")
  @setupMS()
%ELIF (browserType == "NS")
  @setupNS()
%ELSE
  @setupDefault()
%ENDIF
%}
```

MESSAGE Block

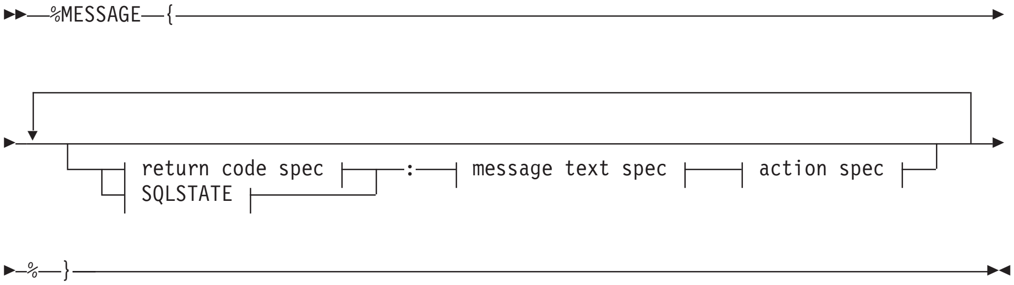
Purpose

Specifies messages to display and actions to take based on the return code from a function.

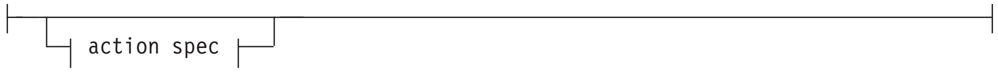
Define the set of return codes, along with their corresponding messages and actions in the MESSAGE block. When a function call completes, Net.Data compares its return code with return codes defined in the MESSAGE block. If the function's return code matches one in the MESSAGE block, Net.Data displays the message and evaluates the action to determine whether to continue processing or exit the Net.Data macro.

A MESSAGE block can be global in scope, or local to a single FUNCTION block. If the MESSAGE block is defined at the outermost macro layer, it is considered global in scope. When multiple global MESSAGE blocks are defined, only the last block processed is considered active. If the MESSAGE block is defined inside a FUNCTION block, the block is local in scope to the FUNCTION block where it is defined. See the MESSAGE block section in the *Net.Data Administration and Programming Guide* for return code processing rules.

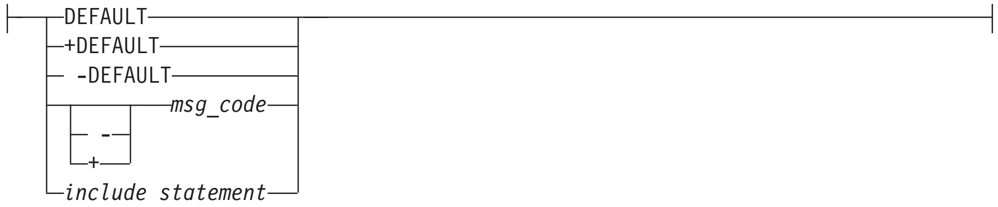
Syntax



action spec



return code spec



SQLSTATE

```
├──SQLSTATE──:──┬─state_id─────────────┬──────────────────────────────────────┤
                └─alphanumeric string──┘
```

message text spec

```
                ┌──────────────────────────┐
                ▼                          │
├──┬─"──────────┬─────────────────────┬────┴──"──┬─────────────────────────────────┤
   │            ├─string──────────────┤          │
   │            ├─variable reference──┤          │
   │            ├─function call───────┤          │
   │            └─(new_line)──────────┘          │
   │            ┌──────────────────────────┐     │
   │            ▼                          │     │
   ├─{──────────┬─────────────────────┬────┴──%}─┤
   │            ├─string──────────────┤          │
   │            ├─variable reference──┤          │
   │            └─function call───────┘          │
   └─include statement───────────────────────────┘
```

action spec

```
         ┌─EXIT──────┐
├──┬─:───┼───────────┼──┬──────────────────────────────────────────────────────┤
   │     └─CONTINUE──┘  │
   └─include statement──┘
```

Values

%MESSAGE

A keyword for the block that defines a set of return codes, the associated messages, and the actions Net.Data takes when a function call is returned.

return code spec

A positive or negative integer. If the value of the Net.Data RETURN_CODE variable matches the *return code spec* value, the remaining information in the message statement is used to process the function call. You can also specify messages for return codes not specifically entered in the MESSAGE block.

+DEFAULT

A keyword used to specify a default positive message code. Net.Data uses the information in this message statement to process the function call if RETURN_CODE is greater than zero (0) and an exact match is not specified.

-DEFAULT

A keyword to specify a default negative message code. Net.Data uses the information in this message statement to process the function call if RETURN_CODE is less than zero (0) and an exact match is not specified.

DEFAULT

A keyword to specify the default message code. Net.Data uses the information in this message statement to process the function call, if all of the following conditions are met:

- If RETURN_CODE is greater or less than zero, but not zero
- If no exact match for the return code is specified
- If the +DEFAULT or -DEFAULT values are not specified for when RETURN_CODE is greater or less than zero

msg_code

The message code that specifies errors and warnings that can occur during processing. A string of numeric digits with values from 0 to 9.

SQLSTATE

A keyword that provides application programs with common codes for common error conditions.The SQLSTATE values are based on the SQLSTATE specification contained in the SQL standard and the coding scheme is the same on all IBM implementations of SQL. **Restriction:** Not supported on the OS/400 platform.

state_id

An alphabetic or numeric string beginning with an alphabetic character or underscore and containing any combination of alphabetic, numeric, or underscore characters.

alphanumeric string

An alphabetic or numeric string containing any combination of alphabetic or numeric characters. It cannot contain punctuation.

message text spec

A string that is sent to the Web browser if the RETURN_CODE matches the *return_code* value in the current message statement.

string

Any sequence of alphabetic and numeric characters and punctuation. If the string appears within double quotes, the new-line character is not allowed.

variable reference

Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

function call

Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function Call (@)” on page 22 for syntax and examples.

action spec

Determines what action Net.Data takes if the RETURN_CODE matches the *return_code* value in the current message statement.

EXIT

A keyword that specifies to exit the macro immediately when the error or warning corresponding to the specified message code occurs. This value is the default.

CONTINUE

A keyword that specifies to continue processing when the error or warning corresponding to the specified message code occurs.

include statement
The INCLUDE statement. Reads and incorporates a file into the Net.Data macro. The INCLUDE statement can appear anywhere in the MESSAGE. See “INCLUDE Statement” on page 32 for syntax and examples.

Context

The MESSAGE block can be found in these contexts:

- FUNCTION block
- IF block
- Outside of all blocks or statements in the declaration part of the Net.Data macro

Restrictions

The MESSAGE block can contain these elements:

- Comment block
- Function calls
- Variable references
- HTML statements
- Strings
- INCLUDE statement

SQLSTATE is not supported on the OS/400 platform.

Examples

Example 1: A local MESSAGE block

```
%MESSAGE{
-601: {<H3>The table has already been created, please go back and enter your name.</H3>
<P><a href="input">Return</a>
%}
default: "<H3>Can’t continue because of error $(RETURN_CODE)</H3>"
%}
```

Example 2: A global MESSAGE block

```
%{ global message block %}
%MESSAGE {
   -100     : "Return code -100 message"   : exit
    100     : "Return code 100 message"    : continue
   +default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %}   : continue
%}

%{ local message block inside a FUNCTION block %}
%FUNCTION(DTW_REXX) my_function() {
  %EXEC { my_command.cmd %}
  %MESSAGE {
     -100     : "Return code -100 message"   : exit
      100     : "Return code 100 message"    : continue
     -default : {
This is a long message that spans more
than one line. You can use HTML tags, including
links and forms, in this message. %}   : exit
  %}
```

Example 3: A MESSAGE block containing INCLUDE statements.

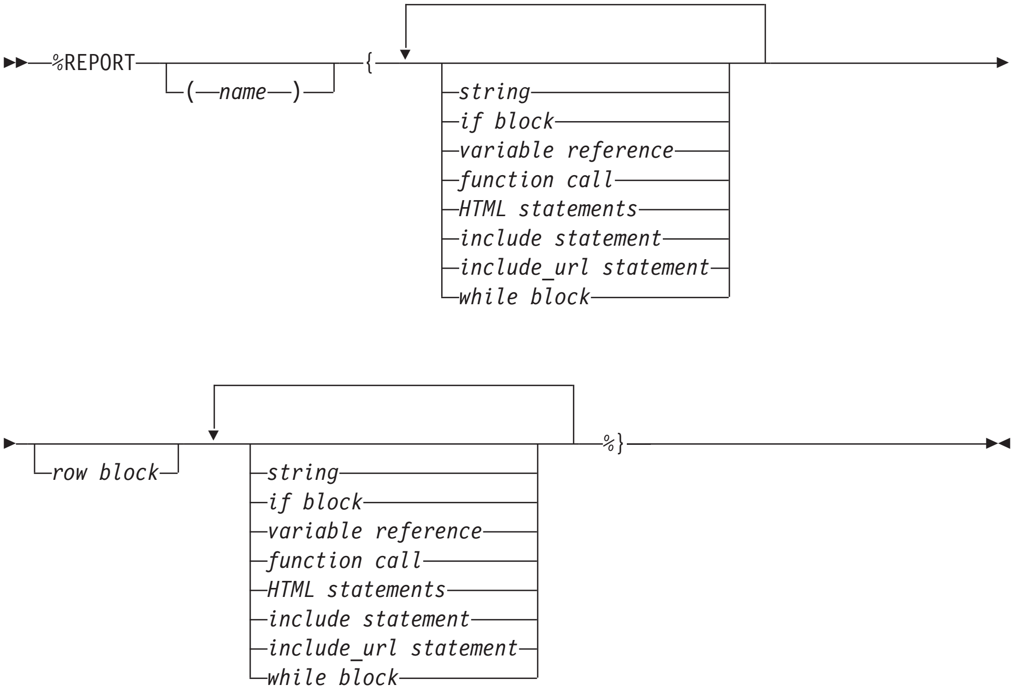
```
%message {
  %include "rc1000.msg"
  %include "rc2000.msg"
  %include "defaults.msg"
%}
```

REPORT Block

Purpose

Formats output from a function call. You can enter a table name parameter to specify that the report is to use the data in the named table. Otherwise, the report is generated with the first output table found in the function parameter list, or with the default table data if no table name is in the list.

Syntax



Values

%REPORT
The keyword for specifying formatting instructions for the output of a function call. You can use header and footer information for the report.

name
An alphabetic or numeric string beginning with an alphabetic character or underscore and containing any combination of alphabetic, numeric, or underscore characters.

string
Any sequence of alphabetic and numeric characters and punctuation.

if block
The IF block. Performs conditional string processing. String values in the condition list are treated as numeric for comparisons if they represent integers and have no leading or trailing white space. They can have one leading plus (+) or minus (-) sign. See “IF Block” on page 26 for syntax and examples.

variable reference
Returns the value of a previously defined variable and is specified with $ and ().

For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function Call (@)” on page 22 for syntax and examples. **Restriction:** The REPORT block cannot include SQL function calls, except in the OS/400 environment.

HTML statements
Includes any alphabetic or numeric characters, as well as HTML tags to be formatted for the client’s browser.

include statement
The INCLUDE statement. Reads and incorporates a file into the Net.Data macro. See “INCLUDE Statement” on page 32 for syntax and examples.

include_url statement
The INCLUDE_URL statement. Reads and incorporates another file into the Net.Data macro in which the statement is specified. The specified file can exist on a local or remote server. See “INCLUDE_URL Statement” on page 34 for syntax and examples.

row block
The ROW block. Displays HTML formatted data once for each row of data that is returned from a function call. See “ROW Block” on page 50 for syntax and examples.

while block
The WHILE block. Performs looping with conditional string processing. See “WHILE Block” on page 54 for syntax and examples.

Context

The REPORT block can be found in these contexts:

- FUNCTION statement or block
- MACRO_FUNCTION block
- SQL statement or block

Restrictions

The REPORT block can contain these elements:

- Comment block
- IF block
- INCLUDE statements
- INCLUDE_URL statements
- ROW blocks
- WHILE blocks
- Function calls

  For OS/390 platform: SQL functions cannot be called from inside SQL functions.
- HTML statements
- Strings
- Variable references

Examples

Example 1: A two-column HTML table showing a list of names and locations

```
%REPORT{
<H2>Query Results</H2>
<P>Select a name for details.
<TABLE BORDER=1>
<TR><TD>Name</TD><TD>Location</TD>
%ROW{
<TR>
<TD>
<a href="/cgi-bin/db2www/name.mac/details?name=$(V1)&location=$(V2)">$(V1)</a></TD>
<TD>$(V2)</TD>
%}
</TABLE>
%}
```

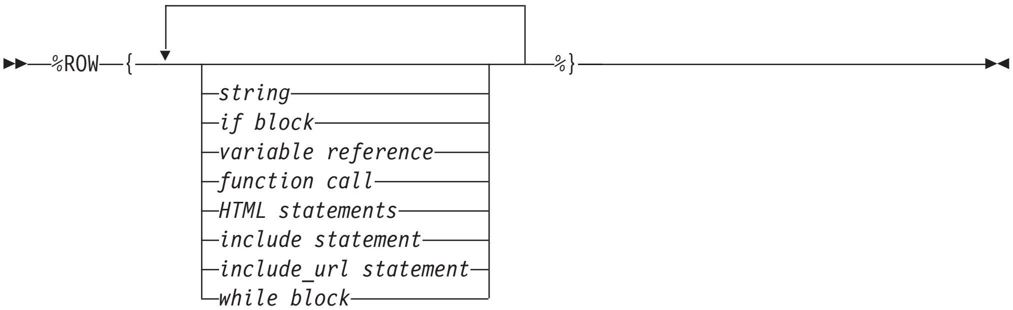
Selecting a name in the table calls the *details* HTML block of the *name.mac* Net.Data macro and sends it the two values as part of the URL. In this example, the values can be used in *name.mac* to look up additional details about the name.

ROW Block

Purpose

Processes each table row returned from a function call. Net.Data processes the statements within the ROW block once for each row.

Syntax



Values

%ROW
The keyword that specifies that HTML formatted data is to be displayed, once for each row of data returned from a function call.

string
Any sequence of alphabetic and numeric characters and punctuation.

if block
The IF block. Performs conditional string processing. String values in the condition list are treated as numeric for comparisons if they are strings that represent integers and have no leading or trailing white space. They can have a single leading plus (+) or minus (-) sign. See “IF Block” on page 26 for syntax and examples.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or built-in functions with specified arguments. See “Function Call (@)” on page 22 for syntax and examples. **Restriction:** ROW cannot include function calls that are SQL function calls, except in the OS/400 environment.

HTML statements
Includes any alphabetic or numeric characters, as well as HTML tags to be formatted for the client’s browser.

include statement
The INCLUDE statement. Reads and incorporates a file into the Net.Data macro. See “INCLUDE Statement” on page 32 for syntax and examples.

include_url statement
The INCLUDE_URL statement. Reads and incorporates another file into the

Net.Data macro in which the statement is specified. The specified file can exist on a local or remote server. See “INCLUDE_URL Statement” on page 34 for syntax and examples.

| **while block**
| The WHILE block. Performs looping with conditional string processing. See
| “WHILE Block” on page 54 for syntax and examples.

Context

The ROW block can be found in these contexts:

- REPORT block

Restrictions

The ROW block can contain these elements:

- Comment block
- IF blocks
- INCLUDE statements
- INCLUDE_URL statements
- WHILE blocks
- Function calls

  For OS/390 platform: SQL functions cannot be called from inside SQL functions.
- Variable references
- HTML statements
- Strings

Examples

Example 1: A two-column HTML table showing a list of names and locations

```
%REPORT{
<H2>Query Results</H2>
<P>Select a name for details.
<TABLE BORDER=1>
<TR><TD>Name</TD><TD>Location</TD>

%ROW{
<TR>
<TD>
<a href="/cgi-bin/db2www/name.mac/details?name=$(V1)&location=$(V2)">$(V1)</a></TD>
<TD>$(V2)</TD>
%}

</TABLE>
%}
```

Selecting a name in the table calls the *details* HTML block of the *name.mac* Net.Data macro and sends it the two values as part of the URL. In this example, the values can be used in *name.mac* to look up additional details about the name.

TABLE Statement

Purpose

Defines a variable which is a collection of related data. It contains an array of identical records, or rows, and an array of column names describing the fields in each row. A table statement can only be in a DEFINE statement or block.

Syntax

```
►►──%TABLE──┤ upper limit ├──────────────────────────────────────────────►◄
```

upper limit

```
├──┬─────────────────────┬──────────────────────────────────────────────┤
   └─(──┬─number─┬──)──┘
        └─ALL────┘
```

Values

%TABLE
A keyword that specifies the definition of a collection of related data containing an array of identical records, or rows, and an array of column names describing the fields in each row.

upper limit
The number of rows that can be contained in the table.

number
A string of digits with values from 0 to 9. A value of 0 allows for unlimited number of rows in the table.

ALL
A keyword that allows for an unlimited number of rows in the table.

Context

The TABLE statement can be found in these contexts:

- DEFINE statement

Restrictions

The TABLE statement can contain these elements:

- Comment block
- Numbers

Examples

Example 1: A Net.Data table with an upper limit of 30 rows

```
%DEFINE myTable1=%TABLE(30)
```

Example 2: A Net.Data table that uses the default of all rows

```
%DEFINE myTable2=%TABLE
```

Example 3: A Net.Data table that specifies all rows

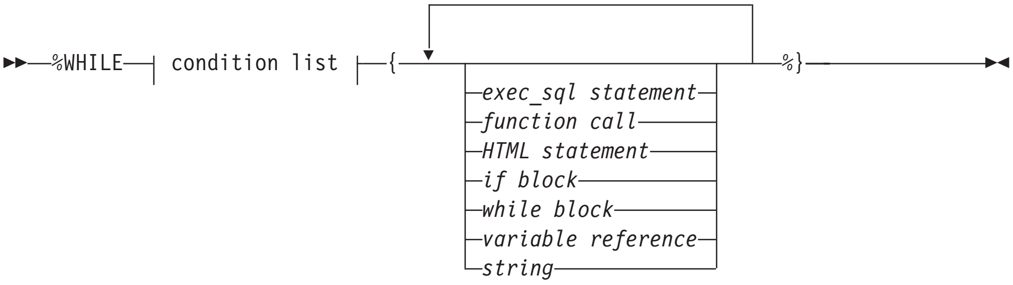
```
%DEFINE myTable3=%TABLE(ALL)
```

WHILE Block

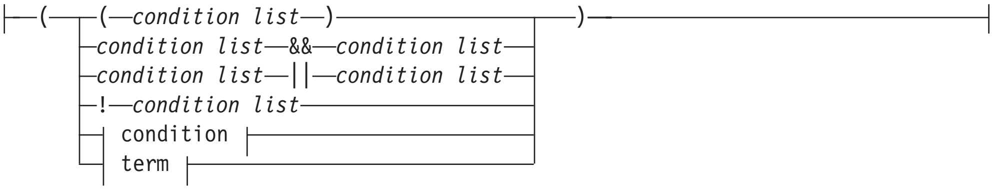
Purpose

Provides a looping construct based on conditional string processing. You can use the WHILE block in the HTML block, the REPORT block, the ROW block, the IF block, and the MACRO_FUNCTION block. String values in the condition list are treated as numeric for comparisons if they are strings that represent integers and have no leading or trailing white space. They can have a single leading plus (+) or minus (-) sign.

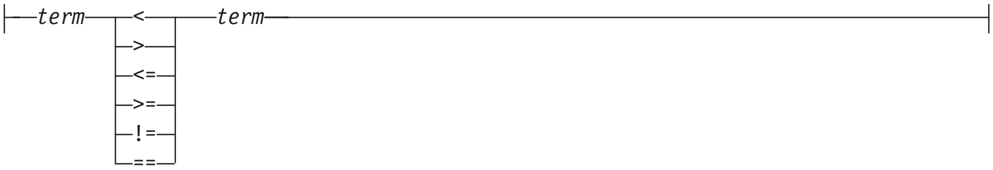
Syntax



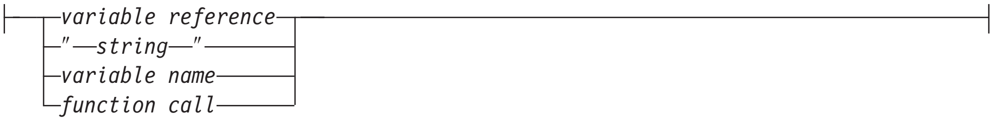
condition list



condition



term



Values

%WHILE
The keyword that specifies loop processing.

condition list
Compares the values of conditions and terms. Condition lists can be connected using Boolean operators. A condition list can be nested inside another condition list.

condition
A comparison between two terms using comparison operators. An IF condition is treated as a numeric comparison if both of the following conditions are true:

- The condition operator is one of the following operators: <,<=,>,>=,==,!=
- Both terms are strings representing valid integers, where a valid integer is a string of digits, optionally proceeded by a plus (+) or minus (-) sign, and no other white space.

If either condition is not true, a normal string comparison is performed.

term
A variable name, string, variable reference, for function call.

exec_sql statement
A DB2WWW Release 1 language element that is supported for compatibility. See “Appendix A. DB2 WWW Connection” on page 223 or DB2 World Wide Web Release 1 documentation.

function call
Invokes one or more previously defined FUNCTION or MACRO_FUNCTION blocks, or built-in functions with specified arguments. See “Function Call (@)” on page 22 for syntax and examples.

HTML statement
Includes any alphabetic or numeric characters, as well as HTML tags to be formatted for the client’s browser.

if block
The IF block. Performs conditional string processing. String values in the condition list are treated as numeric for comparisons if they represent integers and have no leading or trailing white space. They can have one leading plus (+) or minus (-) sign. See “IF Block” on page 26 for syntax and examples.

while block
The WHILE block. Performs looping with conditional string processing. See “WHILE Block” on page 54 for syntax and examples.

variable reference
Returns the value of a previously defined variable and is specified with $ and (). For example: if `VAR='abc'`, then `$(VAR)` returns the value `'abc'`. See “Variable Reference” on page 4 for syntax information.

string
Any sequence of alphabetic and numeric characters and punctuation. A string in the term of the condition list can contain any character except the new-line character.

variable name
One or more names, each additional name concatenated by a period (.). See “Variable Name” on page 4 for syntax information.

Context

The WHILE block can be found in these contexts:

- HTML block
- REPORT block
- ROW block
- MACRO_FUNCTION block
- IF block
- WHILE block

Restrictions

The WHILE block can contain these elements:

- Comment block
- EXEC_SQL statement
- IF block
- WHILE block
- Strings
- HTML statements
- Function calls
- Variable references
- INCLUDE statements

Examples

Example 1: A WHILE block that generates rows in a table

```
%DEFINE loopCounter = "1"

%HTML(build_table) {
%WHILE (loopCounter <= "100") {
  %{ generate table tag and column headings %}
  %IF (loopCounter == "1")
     <TABLE BORDER>
     <TR>
     <TH>Item #
     <TH>Description
     </TR>
  %ENDIF

  %{ generate individual rows %}
  <TR>
  <TD>
  <TD>$(loopCounter)
  <TD>@getDescription(loopCounter)
  </TR>

  %{ generate end table tag %}
  %IF (loopCounter == "100")
     </TABLE>
  %ENDIF

  %{ increment loop counter %}
  @dtw_add(loopCounter, "1", loopCounter)
%}
%}
```

Chapter 2. Variables

Net.Data provides two types of variables: user-defined variables and Net.Data variables.

"User-defined Variables" on page 58

Variables that you define for your application. You can define the variables that perform the following tasks:

- **"Conditional Variables" on page 58**
  Assign a variable value based on the value of another variable or string.
- **"Environment Variables" on page 59**
  Use the ENVVAR language construct to reference environment variables.
- **"Executable Variables" on page 59**
  Use the EXEC language construct to invoke other programs from a variable reference or function with executable variables.
- **"Hidden Variables" on page 61**
  Hide variable reference from HTML source.
- **"List Variables" on page 61**
  Build a delimited string of values using the LIST language construct.
- **"Table Variables" on page 62**
  Pass an array of values to and from a function. Can be used for report output.

Net.Data Variables

Variables that are for miscellaneous processing and file manipulation, table processing, report formatting, and language environments.

Some variables have values that you can define or modify, others are defined by Net.Data. The description for the variable specifies whether you define a value or not. See the description of a variable to determine how the value is defined.

The following variable types are provided by Net.Data:

- **"Net.Data Table Processing Variables" on page 63**
  Defined by Net.Data to let you process Net.Data tales. Use these variables to access data from SQL queries and function calls. They are only recognized inside a REPORT block, unless otherwise specified.
- **"Net.Data Report Variables" on page 73**
  Help you customize reports from a function. You must define these variables before referencing them. You can define or reference report variables in any Net.Data macro block.
- **"Net.Data Language Environment Variables" on page 81**
  Help you customize the way FUNCTION blocks are processed, using language environments.
- **"Net.Data Miscellaneous Variables" on page 99**
  Defined by Net.Data to affect Net.Data processing, find out the status of a function call, and obtain information about the result set of a database query. Some miscellaneous variables are set by Net.Data and cannot be changed.

The output for many Net.Data variables varies depending on the operating system on which it runs.

Constants can be up to 256KB in a Net.Data macro. Thus, you cannot initialize a variable or set a default value whose length is greater than 256 KB in a macro file.

In this chapter, operating system support for each variable is specified. The following list defines operating system abbreviations:

HP-UX	Hewlett Packard UNIX operating system
SCO	Santa Cruz UNIX operating system
SUN	Sun Solaris UNIX operating system
Win NT	Microsoft's Windows NT operating system

User-defined Variables

This section describes the user-defined variables. You define these variables within the macro file.

Conditional Variables

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

The value of a conditional variable is conditionally set based on the value of another variable or string. This is also called a *ternary operation*.

The syntax of conditional variable is:

```
test ? trueValue : falseValue
```

Where:

test Is a condition to test.

trueValue
Is the value to use if the test is true.

falseValue
Is the value to use if the test is false.

Example 1: A conditional variable defined with two possible values

```
varA = varB ? "value_1" : "value_2"
```

If `varB` exists, `varA=value_1`, otherwise `varA=value_2`.

Example 2: A conditional variable defined with a variable reference

```
varname = ? "$(value_1)"
```

In this case, *varname* is null if *value_1* is null, otherwise varname is set to *value_1*.

Example 3: A conditional variable used with a LIST statement and WHERE clause

```
%DEFINE{
%list " AND " where_list
where_list    =  ? "custid = $(cust_inp)"
where_list    =  ? "product_name LIKE '$(prod_inp)%'"
where_clause  =  ? "WHERE $(where_list)"
%}

%FUNCTION(DTW_SQL) mySelect() {
   SELECT * FROM prodtable $(where_clause)
%}
```

Conditional and LIST variables are most effective when used together. The above example shows how to set up a WHERE clause in the DEFINE block. The variables *cust_inp* and *prod_inp* are HTML input variables passed from the Web browser, usually from an HTML form. The variable *where_list* is a LIST variable made of two conditional statements, each statement containing a variable from the Web browser.

If the Web browser returns values for both variables *cust_inp* and *prod_inp*, for example, `IBM` and `755C`, the *where_clause* is:

```
WHERE custid = IBM AND product_name LIKE '755C%'
```

If either variable *cust_inp* or *prod_inp* is null or not defined, the WHERE clause changes to omit the null value. For example, if *prod_inp* is null, the WHERE clause is:

```
WHERE custid = IBM
```

If both values are null or undefined, the variable where_clause is null and no WHERE clause appears in SQL queries containing $(*where_clause*).

Environment Variables

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Environment variables let you use the Net.Data ENVVAR language construct to reference environment variables that exist in the process under which Net.Data is running.

Example 1: A variable is assigned the value of an environment variable

```
%define SERVER_NAME=%ENVVAR

...

The server is $(SERVER_NAME)
```

The environment variable *SERVER_NAME* has the value of the current server name, which, in this example, is `www.software.ibm.com`.

```
The server is www.software.ibm.com
```

See “ENVVAR Statement” on page 13 for more information about the ENVVAR statement.

Executable Variables

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Executable variables allow you to invoke other programs from a variable reference using the executable variable feature. An executable variable is defined in a Net.Data macro using the EXEC language element. For more information about the EXEC language element, see "EXEC Block or Statement" on page 14.

When Net.Data encounters an executable variable in a macro file, it looks for the referenced executable program using the following method:

1. It searches the EXEC_PATH in the Net.Data initialization file. See the configuration chapter in *Net.Data Administration and Programming Guide* for more information about EXEC_PATH.
2. If Net.Data does not locate the program, it searches the directories defined by the system PATH environment variable. If it locates the executable program, Net.Data runs the program.

Example 1: An executable variable definition

```
%DEFINE runit=%exec "testProg"
```

The variable *runit* is defined to execute the executable program *testProg*; *runit* becomes an executable variable.

Net.Data runs the executable program when it encounters a executable variable reference in a Net.Data macro. For example, the program *testProg* is executed when a executable variable reference is made to the variable *runit* in a Net.Data macro.

A simple method is to reference an executable variable from another variable definition. Example 2 demonstrates this method. The variable *date* is defined as an executable variable and *dateRpt* is then defined as a variable reference, that contains the executable variable.

Example 2: An executable variable as a variable reference

```
%DEFINE date=%exec "date"
%DEFINE dateRpt="Today is $(date)"
```

When Net.Data resolves the variable reference $(*dateRpt*), Net.Data searches for the executable date, runs the program, and returns:

```
Today is Tue 11-07-1995
```

An executable variable is never set to the value of the output of the executable program it calls. Using the previous example, the value of date is null. If you use it in a DTW_ASSIGN function call to assign its value to another variable, the value of the new variable after the assignment is null also. The only purpose of an executable variable is to invoke the program it defines.

You can also pass parameters to the program to be executed by specifying them with the program name on the variable definition.

Example 3: Executable variables with parameters

```
%DEFINE mph=%exec "calcMPH $(distance) $(time)"
```

The values of `distance` and `time` are passed to the program `calcMPH`.

Hidden Variables

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

With hidden variables, you can reference variables while hiding the actual variable value in your HTML source. To use hidden variables:

1. Define a variable for each string you want to hide.
2. In the HTML block where the variables are referenced, use double dollar signs instead of a single dollar sign to reference the variables. For example, $$(X) instead of $(X).

Example 1: Hidden variables in a HTML form

```
%HTML(INPUT) {
<FORM ...>
<P>Select fields to view:
<SELECT NAME="Field">
<OPTION VALUE="$$(name)"> Name
<OPTION VALUE="$$(addr)"> Address
.
.
.
</FORM>
%}

%DEFINE{
name="customer.name"
addr="customer.address"
%}

%FUNCTION(DTW_SQL) mySelect() {
  SELECT $(Field) FROM customer
%}
.
.
.
```

When the HTML form is displayed on a Web browser, `$$(`*`name`*`)` and `$$(`*`addr`*`)`are replaced with `$(`*`name`*`)` and `$(`*`addr`*`)` respectively, so the actual table and column names never appear on the HTML form and no one can tell that the true variable names are hidden. When the customer submits the form, the HTML(REPORT) block is called. When `@mySelect()` calls the FUNCTION block, `$(`*`Field`*`)` is substituted in the SQL statement with `customer.name` or `customer.addr` in the SQL query.

List Variables

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

You can use list variables to build a delimited string of values. They are particularly useful in helping you construct an SQL query with multiple items like those found in some WHERE or HAVING clauses.

The blanks are significant. Usually you want to have a blank space on both sides of the value. Most queries use Boolean or mathematical operators (for example, AND, OR, and >). See “LIST Statement” on page 36 for syntax and more information.

Example 1: Use of conditional, hidden, and list variables

```
%HTML(INPUT){
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/example2.max/report">
Select one or more cities:<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$(cond1)">Sao Paulo<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$(cond2)">Seattle<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$(cond3)">Shanghai<BR>
<INPUT TYPE="submit" VALUE="Submit Query">
</FORM>
%}

%DEFINE{
DATABASE="custcity"
%LIST " OR " conditions
cond1="cond1='Sao Paolo'"
cond2="cond2='Seattle'"
cond3="cond3='Shanghai'"
whereClause= ? "WHERE $(conditions)" : ""
%}

%FUNCTION(DTW_SQL) mySelect(){
SELECT name, city FROM citylist
$(whereClause)
%}

%HTML(REPORT){
@mySelect()
%}
```

If no boxes are checked in the HTML form, *conditions* is null, so `whereClause` is also null in the query. Otherwise, `whereClause` has the selected values separated by the Boolean operator OR. For example, if all three cities are selected, the SQL query is:

```
SELECT name, city FROM citylist
WHERE cond1='Sao Paolo' OR cond2='Seattle' OR cond3='Shanghai'
```

Example 2: Value separators

```
%DEFINE %LIST " | " VLIST
%REPORT{
%ROW{
<EM>$(ROW_NUM):</EM> $(VLIST)
%}
%}
```

The table processing variable VLIST uses two quotes and an OR bar, " | ", as a value separator in this example. The string of values are separated by the value in quotes.

Table Variables

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Table variables contain an array of values and the associated column names. Each element in the array is a row. Use table variables to pass groups of values to a function. You can refer to the individual elements of a table (the rows) in a REPORT block of a function. Table variables are often used for output from an SQL function and input to a report, but you can also pass them as IN, OUT, or INOUT parameters to any non-SQL function. Tables can only be passed to SQL functions as OUT parameters. See “TABLE Statement” on page 52 for syntax and more information.

Example 1: A SQL result set that is passed to a REXX program

```
%DEFINE{
DATABASE = "iddata"
MyTable = %TABLE(ALL)
DTW_DEFAULT_REPORT = "NO"
%}

%FUNCTION(DTW_SQL) Query(OUT table) {
select * from survey
%}

%FUNCTION(DTW_REXX) showTable(INOUT table) {
  Say 'Number of Rows: 'table_ROWS
  Say 'Number of Columns: 'table_COLS
  do j=1 to table_COLS
    Say "Here are all of the values for column " table_N.j ":"
    do i = 1 to table_ROWS
       Say "<B>"i"</B>: " table_V.i.j
    end
  end
%}

%HTML (report){
<HTML>
<PRE>
@Query(MyTable)
<p>
@showTable(MyTable)
</PRE>
</HTML>
%}
```

The HTML REPORT block calls an SQL query, saves the result in a table variable and then passes the variable to a REXX function.

Net.Data Table Processing Variables

Net.Data defines these variables for use in the REPORT and ROW blocks, unless noted otherwise. Use these variables to reference values that your queries return.

Restriction: Do not define values for these variables in the DEFINE section.

- "N*n*" on page 64
- "NLIST" on page 65
- "NUM_COLUMNS" on page 66
- "NUM_ROWS" on page 67
- "ROW_NUM" on page 68
- "TOTAL_ROWS" on page 69
- "V_*columnName*" on page 70
- "V*n*" on page 72
- "VLIST" on page 71

N*n*

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The column name returned by a function call or query for column `n`. N*n* is valid in REPORT and ROW blocks.

Net.Data assigns the variable for each column in the table; use the variable in a variable reference, specifying the number of the column you want to reference.

Examples

Example 1: A variable reference for a column name

```
The name of column 2 is $(N2).
```

Example 2: Saves the value of a column name for use outside a REPORT block using DTW_ASSIGN

```
%define col1=""
...
%function (DTW_SQL) myfunc() {
  select * from atable
  %report {
     @dtw_assign(col1, N1)
     %row{ %}
  %}
%}

%html(report) {
@myfunc()
The column name for the first column is $(col1)
%}
```

This example shows how you can use this variable outside the REPORT block by using DTW_ASSIGN. For more information, see “DTW_ASSIGN” on page 151.

Example 3: N*n* within an HTML table to define column names

```
%REPORT{
<H2>Product directory</H2>
<TABLE BORDER=1 CELLPADDING=3>
<TR><TD>$(N1)</TD><TD>$(N2)</TD><TD>$(N5)</TD>
%ROW{
<TR><TD>$(V1)</TD><TD>$(V2)</TD><TD>$(V3)</TD>
%}
</TABLE>

%}
```

NLIST

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Contains a list of all the column names from the result of a function call or query. The default separator is a space.

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Examples

Example 1: A list of column names with ALIGN

```
%DEFINE ALIGN="YES"
...
%FUNCTION (DTW_SQL) myfunc() {
select * from MyTable
%report{
Your query was on these columns: $(NLIST).
%row {
...
%}
%}
%}
```

The list of column names uses a space between column names with ALIGN set to YES.

Example 2: A %LIST variable to change the separator to ″ | ″

```
%DEFINE %LIST " | " NLIST
...
%FUNCTION (DTW_SQL) myfunc() {
select * from MyTable
%report{
Your query was on these columns: $(NLIST).
%row {
...
%}
%}
%}
```

NUM_COLUMNS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The number of table columns that Net.Data is processing in the report block; the columns are returned by a function call or query.

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Examples

Example 1: NUM_COLUMNS used as a variable reference with NLIST

```
%REPORT{
Your query result has $(NUM_COLUMNS) columns: $(NLIST).
...
%}
```

NUM_ROWS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
				X			

Purpose

The number of rows in the table that Net.Data is processing in the REPORT block. The number of rows is affected by the value of the *upper limit* parameter defined for the Net.Data table holding the data. For example, if *upper limit* is set to 30, but the SELECT statement returns 1000 rows, the value of NUM_ROWS is 30. Additionally, if *upper limit* is set to 30 and the SELECT statement returns 20 rows, NUM_ROWS equals 20. See “TABLE Statement” on page 52 for more information about the TABLE statement and the *upper limit* parameter.

NUM_ROWS is not affected by the value of START_ROW_NUM as long as START_ROW_NUM is not passed to the language environment. For example, if START_ROW_NUM is set to 5 (specifying that the table displayed on the Web page should be populated starting with row 5) and the SELECT statement returns 25 rows, NUM_ROWS is set to 25, not 21. The first four rows are discarded from the table, but are included in the value of NUM_ROWS. However, if START_ROW_NUM is passed to the language environment, then NUM_ROWS will only contain the number of rows starting at the row specified by START_ROW_NUM. In the example above, NUM_ROWS will be set to 21.

NUM_ROWS is valid in REPORT and ROW blocks.

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Examples

Example 1: Displays the number of names being processed in the REPORT block

```
%DEFINE DTW_SET_TOTAL_ROWS="YES"

%REPORT{
<H2>E-mail directory</H2>
<UL>
%ROW{
<LI>Name: <a href="mailto:$(V1)">$(V2)</a><BR>
Location: $(V3)
%}
</UL>
Names displayed: $(NUM_ROWS)<BR>
Names found: $(TOTAL_ROWS)
%}
```

ROW_NUM

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

| A table variable whose value Net.Data increments each time a row is processed in a Net.Data table. The variable acts as a counter and its value is the number of the current row being processed.

| This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

RPT_MAX_ROWS can affect the value of ROW_NUM. For example, if 100 rows are in a table, and you have set RPT_MAX_ROWS to `20`, the final value of ROW_NUM is `20`, because row 20 was the last row processed.

ROW_NUM is valid only within a ROW block.

Examples

Example 1: Populates a column in the HTML output by using ROW_NUM to label each row in the table

```
%REPORT{
<TABLE BORDER=1>
<TR><TD> Row Number </TD> <TD> Customer </TD>
%ROW{
<TR><TD> $(ROW_NUM) </TD> <TD> $(V_custname) </TD>
%}
</TABLE>
%}
```

The REPORT block produces a table like the one shown below.

Row Number	Customer
1	Jane Smith
2	Jon Chiu
3	Frank Nguyen
4	Mary Nichols

TOTAL_ROWS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

The total number of rows a query returns, no matter what the value of *upper_limit* for the TABLE language construct. For example, if RPT_MAX_ROWS is set to display a maximum of 20 rows, but the query returns 100 rows, this variable is set to 100 after ROW processing.

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Operating system differences:

- On the OS/400 operating system, this variable can be referenced anywhere in a REPORT or ROW block.
- On the OS/2, Windows NT, and UNIX operating systems, this variable can be reference in the REPORT footer, only.

Required: You must set DTW_SET_TOTAL_ROWS to YES to use this variable. See “DTW_SET_TOTAL_ROWS” on page 91 for more information.

Examples

Example 1: Displays the total number of names found

```
%DEFINE DTW_SET_TOTAL_ROWS="YES"

%REPORT{
<H2>E-mail directory</H2>
<UL>
%ROW{
<LI>Name: <a href="mailto:$(V1)">$(V2)</a><BR>
Location: $(V3)
%}
</UL>
Names displayed: $(NUM_ROWS)<BR>
Names found: $(TOTAL_ROWS)
%}
```

V_*columnName*

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The value for the specified column name for the current row. The variable is not set for undefined column names. A query containing two column names with the same name gives unpredictable results. Consider using an AS clause in your SQL to rename duplicate column names. V_*columnName* is only valid in the ROW block.

| Specify the value of this variable by using it as a variable reference, substituting in
| the actual name of the column.

Values

`V_columnName`

Table 1. V_columnName Values

Values	**Description**
columnName	The column name in current row of the database table.

Examples

Example 1: Using V_*columnName* as a variable reference

```
You have selected $(V_destcity).
```

VLIST

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

| A list of all the field values for the current row being processed in a ROW block.
| VLIST is only valid in a ROW block. The default separator is a space.

| This variable is a predefined variable and its value cannot be modified. Use the
| variable as a variable reference.

Examples

Example 1: Using list tags to display query results

```
%DEFINE ALIGN="YES"

%REPORT{
Here are the results of your query:
<OL>
%ROW{
<LI>$(VLIST)
%}
</OL>
%}
```

Example 2: Using a list variable to change the separator to `<P>`

```
%DEFINE %LIST "<P>" VLIST

%REPORT{
Here are the results of your query:
%ROW{
<HR>$(VLIST)
%}
%}
```

V*n*

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

A field value for each row returned by a function call or SQL query for fields 1 through *n*. V*n* is recognized only in a ROW block.

Net.Data assigns the variable for each field the table; use the variable in a variable reference, specifying the number of the field you want to reference. To use this variable outside the block, assign the value of V*n* to a previously defined global variable or an OUT or INOUT function parameter variable.

Examples

Example 1: Report displaying an HTML table

```
%REPORT{
<H2>E-mail directory</H2>
<TABLE BORDER=1 CELLPADDING=3>
<TR><TD>Name</TD><TD>E-mail address</TD><TD>Location</TD>
%ROW{
<TR><TD>$(V1)</TD>
<TD><a href="mailto:$(V2)">$(V2)</a></TD>
<TD>$(V3)</TD>
%}
</TABLE>
Found $(NUM_ROWS) models matching your description.
%}
```

The second column shows the e-mail address. You can send the person a message by clicking on the link.

Net.Data Report Variables

These variables help you customize your reports. You must define these variables before using them.

- “ALIGN” on page 74
- “DTW_DEFAULT_REPORT” on page 75
- “DTW_HTML_TABLE” on page 76
- “RPT_MAX_ROWS” on page 77
- “START_ROW_NUM” on page 78

ALIGN

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Controls leading and trailing spaces used with the table processing variables NLIST and VLIST. When set to YES, ALIGN provides padding to align table processing variables for display. If you want to embed query results in HTML links or form actions, use the default value of NO to prevent Net.Data from surrounding report variables with leading and trailing spaces.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
ALIGN="YES"|"NO"
```

Table 2. ALIGN Values

Values	Description
YES	Net.Data adds leading and trailing spaces to report variables with spaces to align them for display.
NO	Net.Data does not add leading or trailing spaces. NO is the default.

Examples

Example 1: Using the ALIGN variable to separate each column by a space

```
%DEFINE ALIGN="YES"
<P>Your query was on these columns: $(NLIST)
```

DTW_DEFAULT_REPORT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Determines whether Net.Data generates a default report for functions that have no REPORT block. When this variable is set to YES, Net.Data generates the default report. When set to NO, Net.Data suppresses default report generation. Suppressing the default report is useful, for example, if you receive the results of a function call in a table variable and want to pass the results to a different function to process.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
DTW_DEFAULT_REPORT="YES"|"NO"
```

Table 3. DTW_DEFAULT_REPORT Values

Values	Description
YES	Net.Data generates the default report for functions without REPORT blocks and displays the results at the browser. YES is the default.
NO	Net.Data discards the default report for functions without REPORT blocks.

Examples

Example 1: Overriding the default report generated by Net.Data

```
%DEFINE DTW_DEFAULT_REPORT="NO"
```

DTW_HTML_TABLE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Displays results in an HTML table instead of displaying the table in a text-type format (that is, using the `TABLE` tags rather than the `PRE` tags).

The generated TABLE tag includes a border and cell-padding specification:

```
<TABLE BORDER CELLPADDING=2>
```

| Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
DTW_HTML_TABLE="YES"|"NO"
```

Table 4. DTW_HTML_TABLE Values

Values	Description
YES	Displays table data using HTML table tags.
NO	Displays table data in a text format, using PRE tags. NO is the default.

Examples

Example 1: Displays results from an SQL function with HTML tags

```
%DEFINE DTW_HTML_TABLE="YES"

%FUNCTION(DTW_SQL){
SELECT NAME, ADDRESS FROM $(qTable)
%}
```

RPT_MAX_ROWS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Specifies the number of rows that are displayed in a table generated by a function REPORT block.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

OS/400, Windows NT, OS/2, and UNIX users: To use this variable, ensure that this variable is included as an IN variable in the ENVIRONMENT statement for the database language environments you are using, in the initialization file. See the configuration chapter of *Net.Data Administration and Programming Guide* to learn more about the database language environment statement.

Values

`RPT_MAX_ROWS="ALL"|"0"|"number"`

Table 5. RPT_MAX_ROWS Values

Values	Description
ALL	Indicates that there is no limit on the number of rows to be displayed in a table generated by a function call. All rows will be displayed.
0	Specifies that all rows in the table will be displayed. This value is the same as specifying ALL.
number	A positive integer indicating the maximum number of rows to be displayed in a table generated by a function call. If the FUNCTION block contains a REPORT and ROW block, this number specifies the number of times the ROW block is executed.

Examples

Example 1: Defines RPT_MAX_ROWS in a DEFINE statement

```
%DEFINE RPT_MAX_ROWS="20"
```

The above method limits the number of rows any function returns to 20 rows.

Example 2: Uses HTML input to define the variable with an HTML form

```
Maximum rows to return (0 for no limit):
<INPUT TYPE="text" NAME="RPT_MAX_ROWS" SIZE=3>
```

The lines in the above example can be placed in a FORM tag to let the application users set the number of rows they want returned from a query.

START_ROW_NUM

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Specifies the row number to begin displaying the results of a Net.Data table in a report. Use this variable together with RPT_MAX_ROWS to break queries with large result sets into smaller tables and use a Next button to navigate through the result table.

Restriction: For performance reasons, Net.Data passes START_ROW_NUM to database language environments so that the language environment does not return the entire result set to Net.Data. To pass the variable automatically, include it as an IN variable in the database language environment ENVIRONMENT statement in the initialization file. If this variable is omitted from the ENVIRONMENT statement, the starting row number to be retrieved is assumed to be the first row in the result set.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

`START_ROW_NUM="number"`

Table 6. START_ROW_NUM Values

Values	Description
number	A positive integer indicating the row number with which to begin displaying a report. If START_ROW_NUM is specified in a database language environment’s environment statement in the initialization file, this number specifies the row number of the result set processed by the database language environment. If START_ROW_NUM is not passed to the language environment, this number specifies the row number of the Net.Data table used to display a report.

Examples

Example 1: Scrolling with HTML form Next and Previous buttons

```
%define {
  DTW_HTML_TABLE      = "YES"
  START_ROW_NUM       = "1"
  RPT_MAX_ROWS        = "10"
  totalSize           = ""
  includeNext         = "YES"
  includePrev         = "YES"
  includeLast         = "YES"
  includeFirst        = "YES"
%}

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

%function(DTW_SQL) count(OUT size){
  select count(*) from NETDATADEV.CUSTOMER
  %report{
    %row{
      @DTW_ASSIGN(size,V1)
    %}
  %}
%}

%html(report) {
  %{ get the total number of records if we haven't already %}
  %if (totalSize == "")
    @count(totalSize)
  %endif

  %{ set START_ROW_NUM based on the button user clicked %}
  %if (totalSize <= RPT_MAX_ROWS)
    %{ there's only one page of data %}
    @DTW_ASSIGN(START_ROW_NUM, "1")
    @DTW_ASSIGN(includeFirst, "NO")
    @DTW_ASSIGN(includeLast, "NO")
    @DTW_ASSIGN(includeNext, "NO")
    @DTW_ASSIGN(includePrev, "NO")
  %elif (submit == "First Page" || submit == "")
    %{ first time through or user selected "First Page" button %}
    @DTW_ASSIGN(START_ROW_NUM, "1")
    @DTW_ASSIGN(includePrev, "NO")
    @DTW_ASSIGN(includeFirst, "NO")
  %elif (submit == "Last Page")
    %{ user selected "Last Page" button %}
    @DTW_SUBTRACT(totalSize, RPT_MAX_ROWS, START_ROW_NUM)
    @DTW_ADD(START_ROW_NUM, "1", START_ROW_NUM)
    @DTW_ASSIGN(includeLast, "NO")
    @DTW_ASSIGN(includeNext, "NO")
  %elif (submit == "Next")
    %{ user selected "Next" button %}
    @DTW_ADD(START_ROW_NUM, RPT_MAX_ROWS, START_ROW_NUM)
    %if (@DTW_rADD(START_ROW_NUM, RPT_MAX_ROWS) > totalSize)
      @DTW_ASSIGN(includeNext,"NO")
      @DTW_ASSIGN(includeLast, "NO")
    %endif
  %elif (submit == "Previous")
    %{ user selected "Previous" button %}
    @DTW_SUBTRACT(START_ROW_NUM, RPT_MAX_ROWS, START_ROW_NUM)
    %if (START_ROW_NUM <= "1" )
      @DTW_ASSIGN(START_ROW_NUM,"1")
      @DTW_ASSIGN(includePrev,"NO")
      @DTW_ASSIGN(includeFirst,"NO")
    %endif
  %endif

  %{ run the query to get the data %}
  @myQuery()

  %{ output the correct buttons at the bottom of the report %}
  <center>
  <form method="POST" action="report">
  <input name="START_ROW_NUM" type="hidden" value="$(START_ROW_NUM)">
  <input name="totalSize" type="hidden" value="$(totalSize)">
  %if (includeFirst == "YES" )
  <input name="submit" type="submit" value="First Page">
  %endif
  %if (includePrev == "YES" )
  <input name="submit" type="submit" value="Previous">
  %endif
  %if (includeNext == "YES" )
  <input name="submit" type="submit" value="Next">
  %endif
  %if (includeLast == "YES" )
  <input name="submit" type="submit" value="Last Page">
  %endif
  </form>
  </center>
%}
```

Net.Data Language Environment Variables

Use these variables with functions to help you customize the way FUNCTION blocks are processed by language environments. You might need to define these variables before referencing them.

- “DATABASE” on page 82
- “DB_CASE” on page 84
- “DB2PLAN” on page 85
- “DB2SSID” on page 86
- “DTW_APPLET_ALTTEXT” on page 87
- “DTW_EDIT_CODES” on page 88
- “DTW_MBMODE” on page 89
- “DTW_SAVE_TABLE_IN” on page 90
- “DTW_SET_TOTAL_ROWS” on page 91
- “LOCATION” on page 92
- “LOGIN” on page 93
- “NULL_RPT_FIELD” on page 94
- “PASSWORD” on page 95
- “SHOWSQL” on page 96
- “SQL_STATE” on page 97
- “TRANSACTION_SCOPE” on page 98

DATABASE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Specifies the database or ODBC data source to access when calling a database function. This variable can be changed multiple times within a macro to access multiple databases or ODBC data sources.

OS/400 operating system: This variable is optional. Net.Data, by default, specifies DATABASE="*LOCAL"; the DTW_SQL language environment uses the local relational database directory entry.

Windows NT, OS/2, and UNIX operating systems: Define this variable before calling any database function, except when using the DTW_ORA (Oracle) language environment. Additionally, you must use Live Connection when accessing multiple databases from the same HTML block and through the same language environment.

| Specify the value of this variable using a DEFINE statement or with the
| @DTW_ASSIGN() function.

Values

`DATABASE="`*`dbname`*`"`

Table 7. DATABASE Values

Values	Description
dbname	The name of the database Net.Data connects to.

Examples

Example 1: Specifies to connect to the CELDIAL database for any SQL operations

```
%DEFINE DATABASE="CELDIAL"

%FUNCTION (DTW_SQL) getRpt() {
SELECT * FROM customer
%}

%HTML (report) {
%INCLUDE "rpthead.htm"
@getRpt()
%INCLUDE "rptfoot.htm"
%}
```

The database CELDIAL is accessed when the function `getRpt` is called.

Example 2: Overrides previous DATABASE definitions with DTW_ASSIGN

```
%DEFINE DATABASE="DB2C1"
...
%HTML(monthRpt){
@DTW_ASSIGN(DATABASE, "DB2D1")
%INCLUDE "rpthead.htm"
@getRpt()
%INCLUDE "rptfoot.htm"
%}
```

The HTML block queries the database DB2D1, regardless of what the previous value for DATABASE was.

DB_CASE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

| Specifies which case to use for SQL commands and converts all characters to either upper or lower case. If this variable is not defined, the default action is to not convert the SQL command characters.

| Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
DB_CASE="UPPER"|"LOWER"
```

Table 8. DB_CASE Values

Values	Description
UPPER	Converts all SQL command characters to upper case.
LOWER	Converts all SQL command characters to lower case.

Examples

Example 1: Specifies upper case for all SQL commands

```
%DEFINE DB_CASE="UPPER"
```

DB2PLAN

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
			X				

Purpose

Allocates a plan for a connection to a local DB2 subsystem. The variable specifies the name of a plan for the Net.Data SQL language environment at the local DB2 subsystem that Net.Data will access.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Requirement: This variable must be specified in the Net.Data initialization file on the DTW_SQL ENVIRONMENT statement and optionally in the macro file. An error occurs if the macro attempts to execute an SQL function when this variable is not specified within the Net.Data for OS/390 initialization file or within a macro and not in the initialization file.

Values

```
DB2PLAN="plan_name"
```

Table 9. DB2PLAN Values

Values	Description
plan_name	The name of the DB2 plan. The name can be eight characters or less.

Examples

Example 1: Specifies the plan in the DEFINE statement

```
%DEFINE DB2PLAN="DTWGAV21"
```

DB2SSID

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
			X				

Purpose

Establishes a connection to a local DB2 subsystem. The variable specifies the subsystem ID of the local DB2 subsystem that Net.Data will access. Only one local database connection is allowed for each macro.

| **Requirement:** This variable must be specified in the Net.Data initialization file and optionally in the macro file. An error occurs if the macro attempts to execute an SQL function when this variable is not specified within the Net.Data for OS/390 initialization file and also not defined within a macro.

Values

```
DB2PLAN="subsytem_id"
```

Table 10. DB2SSID Values

Values	Description
subsystem_id	The name of the DB2 subsytem. The name can be eight characters or less.

Examples

Example 1: Specifies a subsystem ID in the DEFINE statement

```
%DEFINE DB2SSID="DBNC"
```

DTW_APPLET_ALTTEXT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X		X	X	X

Purpose

Displays HTML tags and text to browsers that do not recognize the APPLET tag and is used with the the Applet language environment.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
DTW_APPLET_ALTTEXT="HTML_text_and_tags"
```

Table 11. DTW_APPLET_ALTTEXT Values

Values	Description
HTML_text_and_tags	HTML tags and text for browsers that do not recognize the APPLET tag.

Examples

Example 1: Alternate text that indicates a Web browser restriction

```
%DEFINE DTW_APPLET_ALTTEXT="<P>Sorry, your browser is not java-enabled."
```

DTW_EDIT_CODES

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
				X			

Purpose

Converts NUMERIC, DECIMAL, INTEGER and SMALLINT data types that are returned as a result of an SQL operation for the DTW_SQL language environment. The variable DTW_EDIT_CODES is a string of characters that correspond to the resulting columns of the table that DTW_SQL LE will build; for example, the fifth character in DTW_EDIT_CODES will be applied to the fifth column of the result set if that column is one of the supported types. This single character can be any of the supported system supplied edit codes that are defined in *Data Description Specification Reference*.

For example, a DECIMAL(6,0) field would normally be displayed as the character string ’112698’. By specifying an edit code of ’Y’ for that column in the variable DTW_EDIT_CODES, ’Y’ is displayed as a character string that represents the date of ’11/26/98’.

Tip: Applying a user-supplied edit code to a column that results in a character string with non-numeric characters (such as commas or currency symbols) can cause syntax errors if the character string is sent back to the server for subsequent processing within a Net.Data macro. For example, the non-numeric column value might be used for numeric comparisons in subsequent DTW_SQL functions calls, causing syntax errors.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
DTW_EDIT_CODES="edit_code"
```

Table 12. DTW_EDIT_CODES Values

Values	Description
edit_code	Specifies a string of characters that correspond to the resulting columns of the table that the SQL language environment builds.

Examples

Example 1:

```
@DTW_ASSIGN(DTW_EDIT_CODES "JJLJJ*******Y")
```

DTW_MBMODE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X		X	X	X

Purpose

Provides multiple-byte character set (MBCS) support for string and word functions used by the Default language environment. You can set this variable in the Net.Data initialization file, but you can use it in the macro file to set or override the current setting.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

OS/400 users: Net.Data for OS/400 automatically enables functions for MBCS support and does not need this variable. Net.Data for OS/400 ignores this variable in macro files that are migrated to the OS/400 operating system.

Values

```
DTW_MBMODE="YES"|"NO"
```

Table 13. DTW_MBMODE Values

Values	Description
YES	Specifies MBCS support for string and word functions.
NO	Specifies that string and word functions do not have MBCS support. NO is the default.

Examples

```
<DTW_MBMODE="YES"
```

DTW_SAVE_TABLE_IN

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Identifies a table variable that the SQL language environment uses to store table data from a query. This table can then be used later, for example, in a REXX program that analyzes table data.

| Specify the value of this variable using a DEFINE statement or with the
| @DTW_ASSIGN() function.

Values

`DTW_SAVE_TABLE_IN="`*`table_name_var`*`"`

Table 14. DTW_SAVE_TABLE_IN Values

Values	**Description**
table_name_var	The name of a table for the SQL language environment to store table data from a query.

Examples

Example 1: A previously-defined table variable used in a REXX call

```
%DEFINE theTable = %TABLE(2)
%DEFINE DTW_SAVE_TABLE_IN = "theTable"

%FUNCTION(DTW_SQL) doQuery() {
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='MONITOR'
%}

%FUNCTION(DTW_REXX) analyze_table(myTable) {
  %EXEC{ anzTbl.cmd %}
%}

%HTML(doTable) {
@doQuery()
@analyze_table(theTable)
%}
```

A REXX FUNCTION block calls the REXX program `anzTbl.cmd`, which uses the table variable `theTable` to analyze data in the table. The variable `theTable` was returned from a previous SQL function call.

DTW_SET_TOTAL_ROWS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Specifies to a database language environment that the total number of rows in the result set for a query should be assigned to TOTAL_ROWS.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

To pass this variable automatically, include it as an IN variable in the database language environment statement in the Net.Data initialization file. See the configuration chapter of *Net.Data Administration and Programming Guide* to learn more about the database language environment statement.

Values

`DTW_SET_TOTAL_ROWS="YES"|"NO"`

Table 15. DTW_SET_TOTAL_ROWS Values

Values	Description
YES	Assigns the value of the total number of rows to the TOTAL_ROWS variable. **Important:** You must set this value if you want to reference the variable TOTAL_ROWS to determine the number of rows returned from a query.
NO	Net.Data does not set the TOTAL_ROWS variable and TOTAL_ROWS cannot be referenced in a macro file. NO is the default.

Performance tip: Setting DTW_SET_TOTAL_ROWS to YES affects performance because to determine the total rows, the database language environment requires that all rows be retrieved.

Examples

Example 1: Defines DTW_SET_TOTAL_ROWS for using TOTAL_ROWS

```
%DEFINE DTW_SET_TOTAL_ROWS="YES"

...

%FUNCTION (DTW_SQL) myfunc() {
select * from MyTable
%report {
...
%row
...
%}
<P>$(NUM_ROWS) returned. Your query is limited to $(TOTAL_ROWS) rows.
%}
%}
```

LOCATION

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
			X				

Purpose

Establishes a connection to a remote database server. The variable specifies the name by which the remote server is known to the local DB2 subsystem. The value of LOCATION must be defined in the SYSIBM.SYSLOCATIONS table of the Communications Database (CDB). If this variable is not defined within a macro, any SQL requests made by the macro are executed at the local DB2 subsystem.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

`LOCATION="remote_dbase_name"`

Table 16. LOCATION Values

Values	Description
remote_dbase_name	The name of a valid remote database server that is defined in the SYSIBM.SYSLOCATIONS table of the CDB. The name can be eight characters or less.

Examples

Example 1: Defines the remote database location in the DEFINE statement

```
%DEFINE LOCATION="QMFDJ00"
```

LOGIN

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Provides access to protected data by passing a user ID to the database language environment. Use this variable with PASSWORD to incorporate the security algorithms of DB2.

| **Security tip:** While you can code this value in the Net.Data macro, it is preferable
| to have the application user enter user IDs in an HTML form. Additionally, using the
| default value of the Web server ID provides a level of access that might not meet
| your security needs.

| Specify the value of this variable using a DEFINE statement or with the
| @DTW_ASSIGN() function.

Values

`LOGIN="database_user_id"`

Table 17. LOGIN Values

Values	Description
database_user_id	A valid database user ID. The default is to use the user ID that started the Web server.

Examples

Example 1: Restricting access to the user ID, DB2USER

```
%DEFINE LOGIN="DB2USER"
```

Example 2: Using an HTML form input line

```
USERID&#58;  <INPUT TYPE="text" NAME="LOGIN" SIZE=6>
```

This example shows a line you can include as part of an HTML form for application users to enter their user IDs.

NULL_RPT_FIELD

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
				X			

Purpose

Specifies a string the user can provide to the DTW_SQL language environment to represent NULL values that are returned in an SQL result set.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
NULL_RPT_FIELD="null_char"
```

Table 18. NULL_RPT_FIELD Values

Values	**Description**
null_char	Specifies a character to represent NULL values that are returned in an SQL result set. The default is an empty string.

Examples

Example 1: Specifies a string representing NULL values in the SQL language environment

```
%DEFINE NULL_RPT_FIELD = "++++"
```

PASSWORD

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Provides access to protected data by passing a password to the database language environment. Use this variable with LOGIN to incorporate the security algorithms of DB2.

Security tip: While you can code this value in the Net.Data macro, it is preferable to have application users enter passwords in an HTML form.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
PASSWORD="password"
```

Table 19. PASSWORD Values

Values	Description
password	Specifies a valid password to provide automatic access to the database language environment.

Examples

Example 1: Restricting access to application users with the password NETDATA

```
%DEFINE PASSWORD="NETDATA"
```

Example 2: HTML form input line

```
PASSWORD&#58; <INPUT TYPE="password" NAME="PASSWORD" SIZE=8>
```

This example shows a line you can include as part of an HTML form for application users to input their own passwords.

SHOWSQL

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Hides or displays the SQL of the query used on the Web browser. Displaying the SQL during testing is especially helpful when you are debugging your Net.Data macros.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

```
SHOWSQL="YES"|"NO"
```

Table 20. SHOW_SQL Values

Values	**Description**
YES	Displays the SQL of the query sent to the database.
NO	Hides the SQL of the query sent to the database. NO is the default.

Examples

Example 1: Displays all SQL queries

```
%DEFINE SHOWSQL="YES"
```

Example 2: Specifying whether to display SQL using HTML form input

```
SHOWSQL: <INPUT TYPE="radio" NAME="SHOWSQL" VALUE="YES"> Yes
         <INPUT TYPE="radio" NAME="SHOWSQL" VALUE="" CHECKED> No
```

SQL_STATE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Accesses or displays the SQL state value returned from the database.

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Examples

Example 1: Displays the SQL state in the REPORT block

```
%FUNCTION (DTW_SQL) val1() {
 select * from customer
%REPORT {
 ...
%ROW {
 ...
%}
 SQLSTATE=$(SQL_STATE)
%}
```

TRANSACTION_SCOPE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Specifies the transaction scope for SQL commands, determining whether Net.Data issues a COMMIT after each SQL command or after all SQL commands in an HTML block complete successfully. When you specify that all SQL commands must complete successfully before a commit, an unsuccessful SQL command causes all previously executed SQL to *the same database* in that block to be rolled back.

Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Consistency considerations: On operating systems other than OS/400 and OS/390, updates to the database receiving unsuccessful responses might be rolled back while the updates to the other databases accessed in the same HTML block might be committed when all of the following conditions are true:
- `TRANSACTION_SCOPE = "MULTIPLE"` is specified
- Multiple databases are accessed in one HTML block (which is possible when using Live Connection)
- An unsuccessful response is returned from an SQL request

If you access multiple databases from Net.Data using IBM's DataJoiner, you can achieve multiple database update coordination and consistency when updating from Net.Data.

On OS/400 and OS/390, `TRANSACTION_SCOPE = "MULTIPLE"` causes all IBM database updates issued from a single HTML block to be committed or rolled back together.

On operating systems other than OS/400, the REXX, Perl, and Java language environments run in their own separate operating system processes. Thus, any database updates you issue from these language environments are committed or rolled back separately from database updates issued from a Net.Data macro file, regardless of the Net.Data TRANSACTION_SCOPE value.

Values

```
TRANSACTION_SCOPE="SINGLE"|"MULTIPLE"
```

Table 21. TRANSACTION_SCOPE Values

Values	Description
SINGLE	Net.Data issues a COMMIT after each SQL command in an HTML block successfully completes.
MULTIPLE	Specifies the Net.Data issues a COMMIT only after all SQL commands in an HTML block complete successfully. MULTIPLE is the default.

Examples

Example 1: Specifies to issue a COMMIT after each transaction

```
%DEFINE TRANSACTION_SCOPE="SINGLE"
```

Net.Data Miscellaneous Variables

These variables are Net.Data-defined variables that you can use to affect Net.Data processing, find out the status of a function call, and obtain information about the result set of a database query, as well as determine information about file locations and dates. You might find these variables useful in functions you write or use them when testing your Net.Data macros.

- “DTW_CURRENT_FILENAME” on page 100
- “DTW_CURRENT_LAST_MODIFIED” on page 101
- “DTW_DEFAULT_MESSAGE” on page 102
- “DTW_LOG_LEVEL” on page 103
- “DTW_MACRO_FILENAME” on page 104
- “DTW_MACRO_LAST_MODIFIED” on page 105
- “DTW_MP_PATH” on page 106
- “DTW_MP_VERSION” on page 107
- “DTW_PRINT_HEADER” on page 108
- “DTW_REMOVE_WS” on page 109
- “RETURN_CODE” on page 110

DTW_CURRENT_FILENAME

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The name and extension of the current input file. The input file is either a Net.Data macro or a file specified in an INCLUDE statement.

| This variable is a predefined variable and its value cannot be modified. Use the
| variable as a variable reference.

Examples

```
<P>This file is <I>$(DTW_CURRENT_FILENAME)</I>,
and was updated on $(DTW_CURRENT_LAST_MODIFIED).
```

DTW_CURRENT_LAST_MODIFIED

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The date and time the current file was last modified. The current file can be a Net.Data macro file or a file specified in an INCLUDE statement. The output format is determined by the system on which Net.Data runs.

| This variable is a predefined variable and its value cannot be modified. Use the
| variable as a variable reference.

Examples

```
<P>This file is <I>$(DTW_CURRENT_FILENAME)</I>,
and was updated on $(DTW_CURRENT_LAST_MODIFIED).
```

DTW_DEFAULT_MESSAGE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
				X			

Purpose

Contains the message text returned from a call to a built-in function or to language environment when an error occurs.

You can use the DTW_DEFAULT_MESSAGE variable in any part of the Net.Data macro file.

This variable is a predefined variable, it is not recommended to modify its value. Use the variable as a variable reference.

Examples

Example 1: A message stating whether the function completed successfully

```
@function1()
%IF ("$(RETURN_CODE)" == "0")
 The function completed successfully.
%ELSE
The function failed with the return code $(RETURN_CODE).  The error message
   returned is "$(DTW_DEFAULT_MESSAGE)".
%ENDIF
```

Example 2: The default text for when a function returns a non-zero return code

```
%MESSAGE{
default: "<h2>Net.Data received return code: $(RETURN_CODE).
  Error message is $(DTW_DEFAULT_MESSAGE)</h2>" : continue
%}
```

The user sees the default error message, if a function returns a return code other than 0.

DTW_LOG_LEVEL

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X			X	X	X

Purpose

The level of messages that Net.Data writes to the log file.

You can specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Requirement: Define DTW_LOG_DIR in the Net.Data initialization file to initiate logging; otherwise Net.Data does not log messages when you specify the DTW_LOG_LEVEL variable in the macro file.

Values

```
DTW_LOG_LEVEL="OFF|ERROR|WARNING"
```

Table 22. DTW_LOG_LEVEL Values

Values	Description
OFF	Net.Data does not log errors. OFF is the default.
ERROR	Net.Data logs error messages.
WARNING	Net.Data logs warnings, as well as error messages.

Examples

```
%DEFINE DTW_LOG_LEVEL="ERROR"
```

DTW_MACRO_FILENAME

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The name and extension of the current Net.Data macro file.

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Examples

```
<P>This Net.Data macro is <I>$(DTW_MACRO_FILENAME)</I>,
and was updated on $(DTW_MACRO_LAST_MODIFIED).
```

DTW_MACRO_LAST_MODIFIED

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The date and time the Net.Data macro was last modified. The output format depends on the system on which Net.Data runs.

| This variable is a predefined variable and its value cannot be modified. Use the
| variable as a variable reference.

Examples

```
<P>This Net.Data macro is <I>$(DTW_MACRO_FILENAME)</I>,
and was updated on $(DTW_MACRO_LAST_MODIFIED).
```

DTW_MP_PATH

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The path and name of the Net.Data executable file. Depending on your system, the output looks like the following sample path and name:

```
/usr/lpp/internet/server_root/cgi-bin/db2www
```

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Examples

```
The Net.Data executable file is $(DTW_MP_PATH).
```

DTW_MP_VERSION

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The version and release number of Net.Data running on the server. The output is in the following format:

```
Net.Data Version 2.1
```

This variable is a predefined variable and its value cannot be modified. Use the variable as a variable reference.

Examples

```
This Web application uses $(DTW_MP_VERSION).
```

DTW_PRINT_HEADER

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Specifies text for the file header.

You must have this variable set before Net.Data processes any text sent to the Web browser, because Net.Data reads this variable once before displaying text and does not look at it again. Any changes to the DTW_PRINT_HEADER variable are ignored after Net.Data has sent text to the browser.

| **OS/390 users:** If you are using DTW_PRINT_HEADER to generate your own headers (`DTW_PRINT_HEADER="NO"`), you must set `DTW_REMOVE_WS="NO"`.

| Specify the value of this variable using a DEFINE statement or with the @DTW_ASSIGN() function.

Values

`DTW_PRINT_HEADER="YES"|"NO"`

Table 23. DTW_PRINT_HEADER Values

Values	Description
YES	Net.Data prints out the text `Content-type: text/html` for the HTTP header. YES is the default.
NO	Net.Data does not print out an HTTP header. You can generate custom HTTP header information.

Examples

One of the most common uses of this variable is to enable Net.Data macros to send cookies. To set a cookie, the DTW_PRINT_HEADER variable must be set to `NO`, and the first three lines must be the Content-type header, the Set-Cookie statement, and a blank line.

Example 1: Enabling Net.Data to send 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=/

<P>
Any text
%}
```

DTW_REMOVE_WS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Reduces the size of a dynamically generated Web page by compressing extra space caused by tabulators, white space, and new-line characters.

Specify the value of this variable in the DEFINE block.

Using <PRE></PRE> tags: Defining this variable to YES affects the amount and type of white space that is printed. If the variable is set to YES, portions of HTML pages that use <PRE></PRE> tags might not displays as intended.

OS/390 users:

1. If you are using DTW_PRINT_HEADER to generate your own headers (`DTW_PRINT_HEADER="NO"`), you must set `DTW_REMOVE_WS="NO"`.
2. Set this variable in the Net.Data initialization file to specify a value for all of your macros. You can override the value by defining it in the macro file. If DTW_REMOVE_WS is not defined in the macro file, it uses the value in the initialization file.

Values

```
DTW_REMOVE_WS="YES"|"NO"
```

Table 24. DTW_REMOVE_WS Values

Values	Description
YES	Net.Data compresses a sequence of two or more white spaces to one new-line character, generating shorter HTML result pages.
NO	Net.Data does not compress white spaces. NO is the default.

Examples

Example 1: Compressing white space

```
DTW_REMOVE_WS="YES"
```

RETURN_CODE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

The return code returned by a call to a built-in function or a call to a language environment. Net.Data uses this value to process MESSAGE blocks. You can use this variable to determine whether a function call succeeded or failed. A value of zero indicates successful completion of a function call.

You can reference the RETURN_CODE variable in any part of the Net.Data macro file.

| This value is predefined; it is not recommended to modify the value. Use it as a variable reference.

Examples

Example 1: A message stating whether the function completed successfully

```
@function1()
%IF ("$(RETURN_CODE)" == "0")
 The function completed successfully.
%ELSE
The function failed with the return code $(RETURN_CODE).
%ENDIF
```

Example 2: A default message when a return code is not 0

```
%MESSAGE{
default: "<h2>Net.Data received return code: $(RETURN_CODE)</h2>" : continue
%}
```

If a function returns a return code other than 0, the default message is displayed.

Chapter 3. Net.Data Built-in Functions

Net.Data provides a wide variety of functions that you can use without creating your own FUNCTION blocks. Net.Data built-in functions are divided into the following categories:

- **General-purpose functions** help you develop Web pages with Net.Data and do not fit in the other categories. See "General Functions" on page 112.
- **Math functions** perform mathematical operations. See "Math Functions" on page 139.
- **String-manipulation functions** modify strings and characters. See "String Functions" on page 150.
- **Word-manipulation functions** modify words or sets of words. See "Word Functions" on page 166.
- **Table-manipulation functions** help you generate forms and reports from your table data. See "Table Functions" on page 174.
- **Flat-file interface functions** perform file input and output. See "Flat File Interface Functions" on page 193.
- **Web-registry functions** perform operations on a Web registry. See "Web Registry Functions" on page 212.

In the descriptions that follow, function parameters are described as being of type *string*, *integer*, *float*, and *table*. All Net.Data variables are of type string, but the terms integer and float are used to denote a string that represents an integer or float value, respectively.

Function Names

Net.Data built-in functions begin with `DTW_`, which is a reserved prefix. User-defined functions should not use this prefix.

Built-in functions names are not case sensitive.

Input and Output Parameters

Functions can have parameter passing specifications that determine whether Net.Data uses the parameter for input, output, or both input and output. These parameter passing specifications are specified by the following keywords, :

IN Specifies that the parameter passes input data to the language environment from Net.Data.

OUT Specifies that the parameter returns output data from the language environment to Net.Data.

INOUT
Specifies that the parameter passes input data to the language environment and returns output data from the language environment to Net.Data.

Function Result Formatting

Many functions have one or more of the following forms:

- Functions beginning with DTW_r, DTWF_r, and DTWR_r return their results to the function call, so they do not have an output parameter. This example shows the server time:

  ```
  Current local time is @DTW_rTIME().
  ```
- Functions beginning with DTW_m perform the function on multiple parameters. Each parameter behaves as both an input parameter and an output parameter. The function is performed on the parameter and the results are returned in the parameter. This example converts the three input parameters to all capital letters for a consistent look in the display:

  ```
  @DTW_mUPPERCASE(model, style, shipNo)
  Shipment $(shipNo) contains $(quantity) of model $(model) $(style).
  ```
- Other functions beginning with DTW_, DTWF_, and DTWR_ return their results in an output parameter. You must specify the output parameter. This example shows the server time:

  ```
  @DTW_TIME(nowTime)
  Current local time is $(nowTime).
  ```

Function Parameter Rules

Place function parameters in the correct order. You can specify all *input* parameters before the last input parameter can be specified, or specify a null (“”) to accept the default. For example, you can call DTW_TB_INPUT_TEXT as in the following example:

```
@DTW_TB_INPUT_TEXT(myTable, "1", "2", "", "", "32")
```

In the above example the fourth and fifth parameters use default values. Include them as nulls to indicate that “32” is the value for MAXLENGTH in the generated HTML. The final parameter is not specified, so the default value is used. If you choose to accept the default value for MAXLENGTH and the two previous parameters, omit them, as shown below:

```
@DTW_TB_INPUT_TEXT(myTable, "1", "2")
```

You must specify intermediate null values in the parameter lists for input parameters when subsequent non-null *input* parameters exist. You do not need to specify intermediate null input parameters before specifying your final *output* parameter.

General Functions

General functions help you develop Web pages with Net.Data and do not fit in the other categories. The following functions are general-purpose functions:

- “DTW_ADDQUOTE” on page 114
- “DTW_CACHE_PAGE” on page 116
- “DTW_DATE” on page 119
- “DTW_EXIT” on page 120
- “DTW_GETCOOKIE” on page 121
- “DTW_GETENV” on page 123

- “DTW_GETINIDATA” on page 124
- “DTW_HTMLENCODE” on page 125
- “DTW_QHTMLENCODE” on page 127
- “DTW_SENDMAIL” on page 128
- “DTW_SETCOOKIE” on page 132
- “DTW_SETENV” on page 135
- “DTW_TIME” on page 136
- “DTW_URLESCSEQ” on page 137

DTW_ADDQUOTE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Replaces single quotes in an input string with two single quotes. The replacement is needed so that SQL statements can be processed correctly when a string contains a single quote.

Consider using this function for all SQL INPUT statements. For example, if you enter O'Brien as a last name, as in the following example, the single quote might give you an error:

```
INSERT INTO USER1.CUSTABLE (LNAME, FNAME)
VALUES ('O'Brien', 'Patrick')
```

Using the DTW_ADDQUOTE function changes the SQL statement and prevents the error:

```
INSERT INTO USER1.CUSTABLE (LNAME, FNAME)
VALUES ('O''Brien', 'Patrick')
```

Format

@DTW_ADDQUOTE(stringIn, stringOut)

@DTW_rADDQUOTE(stringIn)

@DTW_mADDQUOTE(stringMult, stringMult2, ..., stringMultn)

Values

Table 25. DTW_ADDQUOTE Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string. DTW_mADDQUOTE can have multiple input strings.
string	*stringOut*	OUT	A variable that contains the modified form of *stringIn*.
string	*stringMult*	INOUT	• On input: A variable that contains a string. • On output: A variable containing the input string with each single quote (') character replaced by two single-quote characters.

Examples

Example 1: Adds an extra single quote on the OUT parameter

`@DTW_ADDQUOTE(string1,string2)`

- Input: `string1="John's Web page"`
- Returns: `string2="John''s Web page"`

Example 2: Adds an extra single quote on the returned value of the function call

`@DTW_rADDQUOTE("The title of the article is 'Once upon a time'")`

- Returns: `"The title of the article is ''Once upon a time''"`

Example 3: Adds extra single quotation marks on each of the INOUT parameters of the function call

```
@DTW_mADDQUOTE(string1,string2)
```

- Input: `string1="Joe's bag", string2="'to be or not to be'"`
- Returns: `string1="Joe''s bag", string2="''to be or not to be''"`

Example 4: Inserts extra single quotation marks into data being inserted in a DB2 table

```
%FUNCTION(DTW_SQL) insertName(){
INSERT INTO USER1.CUSTABLE (LNAME,FNAME)
VALUES ('@DTW_rADDQUOTE(lastname)', '@DTW_rADDQUOTE(firstname)')
%}
```

- Input: `lastname="O'Brien", firstname="Patrick"`
- Returns: `"O''Brien", "Patrick"`

DTW_CACHE_PAGE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X							

Purpose

Begins caching all HTML output following the function’s position in the macro file. When invoked, this function attempts to retrieve the specified page from the cache and to send it to the Web browser as if it were the output page generated from the macro. If the page is found and it has not expired, Net.Data stops processing the macro, exits from the macro file, and sends the cached page to the Web browser.

If the requested page is not in the cache or the existing cached page is older than the value of *age*, Net.Data generates a new output page. When the macro successfully completes, Net.Data sends the new page to the browser and caches the page.

Determining the location of the DTW_CACHE_PAGE function in the macro file:

- For most caching applications, specify DTW_CACHE_PAGE at the top of the macro to cache all HTML output. This technique makes it easier to maintain the macro file when new report blocks are added. For example, when the function is in the middle of the macro, it might not be noticed when a HTML report section is added earlier in the macro. Net.Data would not cache the new report output. Additionally, this method improves performance as Net.Data stops all further processing when it determines that the page is cached.
- For advanced caching applications, you can place the function in the HTML output sections when you need to make the decision to cache at a specific point during processing, rather than at the beginning of the macro file. For example, you might need to make the caching decision based on how many rows are returned from a query or function call.

Format

```
@DTW_CACHE_PAGE(cacheid, url, age, status)
```

Values

Table 26. DTW_CACHE_PAGE Parameters

Parameter	Use	Description
cache_id	IN	A string variable identifying the cache where the page will be placed.
cached_page_ID	IN	A string variable containing an identifier used to locate the cached page in a subsequent DTW_CACHE_PAGE cache request. The string can be a URL.
age	IN	A string variable containing a length of time in seconds. This parameter determines whether a page has expired. If the page is older than *age*, the page is not sent to the browser. If *age* is specified as -1, and the page exists in the cache, Net.Data sends it to the Web browser regardless of its age directly from the cache. Net.Data does not replace the page in the cache.

Table 26. DTW_CACHE_PAGE Parameters (continued)

Parameter	Use	Description
status	OUT	A string variable indicating the state of the cached page. Possible values are in lowercase: • **ok:** The output page will be cached when the macro execution terminates. • **new:** The page is not in the cache. • **renew:** The page is in the cache, but has expired. • **no_cache:** The cache identifier specified does not exist. It must be defined in the cache configuration files. Your macro can continue executing without page caching. • **inactive:** The cache you specified has been marked inactive. Your macro can continue executing without page caching. • **busy:** Your macro has issued the DTW_CACHE_PAGE built-in function before in this execution. Your macro can continue executing. • **error:** An error occurred while trying to communicate with the cache.

Examples

Example 1: Places the DTW_CACHE_PAGE function at the beginning of the macro file to capture all HTML output

```
%IF (customer_status == "Classic")
@DTW_CACHE_PAGE("mymacro.mac", "http://www.mypage.org", "-1", status)
%ENDIF
 % DEFINE { ...%}

...

%HTML(OUTPUT) {
 <title>This is the page title
 </head>
 <body>
 <center>
 This is the Main Heading
 <p>It is $(time). Have a nice day!
 </body>
 </html>

%}
```

Example 2: Places the function in the HTML block because the decision to cache depends on the expected size of the HTML output

```
%DEFINE { ...%}

...

%FUNCTION(DTW_SQL) count_rows(){
  select count(*) from customer
%REPORT{
 %ROW{
  @DTW_ASSIGN(ALL_ROWS, V1)
%}
%}
%}

%FUNCTION(DTW_SQL) all_customers(){
  select * from customer
%}

%HTML(OUTPUT) {
<html>
<head>
 <title>This is the customer list
 </head>
 <body>

@count_rows()

 %IF ($(ALL_ROWS) > "100")
 @DTW_CACHE_PAGE("mymacro.mac", "http://www.mypage.org", "-1", status)
%ENDIF

@all_customers()

 </body>
 </html>
%}
```

In this example, the page is cached or retrieved based on the expected size of the HTML output. HTML output pages are considered cache-worthy only when the database table contains more than 100 rows. Net.Data always sends the text in the OUTPUT block, `This is the customer list`, to the browser after executing the macro; the text is never cached. The lines following the function call, `@count_rows()`, are cached or retrieved when the conditions of the IF block are satisfied. Together, both parts form a complete Net.Data output page.

Example 3: Dynamically retrieves the cache ID and the cached page ID

```
%HTML(OUTPUT) {
 %IF (customer == "Joe Smith")

@DTW_CACHE_PAGE(@DTW_rGETENV("DTW_MACRO_FILENAME"), @DTW_rGETENV("URL"),"-1", status)

%ENDIF

...

<html>
 <head>
 <title>This is the page title</title>
 </head>
 <body>
 <center>
 <h3>This is the Main Heading</h3>
 <p>It is @DTW_rDATE(). Have a nice day!
 </body>
</html>

%}
```

DTW_DATE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the current system date in the specified format.

Format

@DTW_DATE(format, stringOut)
@DTW_DATE(stringOut)
@DTW_rDATE(format)
@DTW_rDATE()

Values

Table 27. DTW_DATE Parameters

Data Type	Parameter	Use	Description
string	*format*	IN	A variable or literal string specifying the data format. Valid formats include: D - Day of the year (001–366) E - European date format (dd/mm/yy) N - Normal date format (dd mon yyyy) O - Ordered date format (yy/mm/dd) S - Standard date format (yyyymmdd) U - USA date format (mm/dd/yy) The default is N.
string	*stringOut*	OUT	A variable that contains the date in the specified format.

Examples

Example 1: Normal date format

```
@DTW_DATE(results)
```

- Returns: `results = "25 Apr 1997"`

Example 2: European date format

```
@DTW_DATE("E", results)
```

- Returns: `results="25/04/97"`

Example 3: US date format

```
%HTML(report){
<P>This report created on @DTW_rDATE("U").
```

- Returns: `04/25/97`

DTW_EXIT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X			X	X	X

Purpose

Specifies to leave the macro immediately. Net.Data ensures that the page the macro created so far will be sent to the browser.

Performance tip: Use DTW_EXIT to stop the processing of a macro file when output has been generated in order to save the time Net.Data has to process the entire file.

Format

@DTW_EXIT()

Examples

Example 1: Exiting a macro

```
%HTML(cache_example) {

<html>
 <head>
 <title>This is the page title</title>
 </head>
 <body>
 <center>
 <h3>This is the Main Heading</h3>
 <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
 <! Joe Smith sees a very short page                    !>
 <!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!>
 %IF (customer == "Joe Smith")

@DTW_EXIT()

%ENDIF

...

</body>
 </html>
 %}
```

DTW_GETCOOKIE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X			X	X	X

Purpose

Specifies the name of a cookie to be read and returns the value of the cookie.

To retrieve a cookie, it must have been defined with the DTW_SETCOOKIE() function. See “DTW_SETCOOKIE” on page 132 to learn how to define a cookie.

Tip: Define and retrieve a cookie in two separate HTTP requests. Because a cookie is visible only after it has been sent to the client, if a macro tries to get a cookie that was defined in the same HTTP request, you might receive unexpected results.

Format

@DTW_GETCOOKIE(IN cookie_name, OUT cookie_value)

@DTW_rGETCOOKIE(IN cookie_name)

Values

Table 28. DTW_GETCOOKIE Parameters

Data Type	Parameter	Use	Description
string	*cookie_name*	IN	A variable or literal string that specifies the name of the cookie.
string	*cookie_value*	OUT	A variable containing the value of the cookie retrieved by the function, such as user state information.

Examples

Example 1: Retrieves cookies that contain user ID and password information

```
@DTW_GETCOOKIE("mycookie_name_for_userID", userID)
@DTW_GETCOOKIE("mycookie_name_for_password", password)
```

Example 2: Determines if a cookie for a user exists before gathering user information

```
%HTML(welcome) {
  <html>
  <body>
  <h1>Net.Data Club</h1>
  @DTW_GETCOOKIE("NDC_name", name)
  %IF ($(RETURN_CODE) == "8000") %{ The cookie is not found. %}
  <form method="post" action="remember">
  <p>Welcome to the club. Please enter your name.<br>
  <input name="name">
  <input type="submit" value="submit"><br>
  </form>
  %ELSE
  <p>Hi, $(name). Welcome back.
  %ENDIF
  </body>
  </html>
  %}
```

The HTML welcome section checks whether the cookie `NDC_name` exists. If the cookie exists, the browser displays a personalized greeting. If the cookie does not exist, the browser prompts for the user’s name, and posts it to the HTML remember section, which sets the user’s name into the cookie NDC_name as shown below:

```
%HTML(remember) {
  <html>
  <body>
  <H1>Net.Data Club</H1>
  @DTW_SETCOOKIE("NDC_name", name, "expires=Wednesday, 01-Dec-2010 00:00:00;path=/")
  <p>Thank you.
  <p><a href="welcome">Come back</a>
  </body>
  </html>
  %}
```

Return Codes

If the cookie is not found, return code 8000 is returned. The cookie might not be found for the following reasons:

- The cookie has never been set.
- The cookie has expired.
- The cookie does not have an expiration date and is therefore not persistent; the Web browser that received the cookie exited or was killed.
- The cookie was set with a secure option, and the current HTTP request was sent over an insecure channels.
- The Web browser did not accept the cookie or it did not execute JavaScript programs at the time when the set cookie request was submitted.
- The cookie has been deleted by the Web browser. This can happen when the number of cookies exceeds the limitations of the browser. The limitations are described in Netscape’s specification and at the time of publication are:
  - 300 total cookies
  - 4 kilobytes per cookie, where the name and the value combine to form the 4 kilobyte limit.
  - 20 cookies per server or domain. (Note that completely specified hosts and domains are treated as separate entities and have a 20 cookie limitation for each, not combined)

  Servers should not expect clients to be able to exceed these limits. When the 300 cookie limit or the 20 cookie per server limit is exceeded, clients should delete the least recently used cookie. When a cookie larger than 4 kilobytes is encountered the cookie should be trimmed to fit, but the name should remain intact as long as it is less than 4 kilobytes.

  See Netscape’s specification for the latest information in “Persistent Client State HTTP Cookies,” which is available at:

  `http://search.netscape.com/newsref/std/cookie_spec.html`

DTW_GETENV

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the value of the specified environment variable. You can also use ENVVAR to reference the values of environment variables. For more information, see “ENVVAR Statement” on page 13.

Format

@DTW_GETENV(envVarName, envVarValue)

@DTW_rGETENV(envVarName)

Values

Table 29. DTW_GETENV Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*envVarName*	IN	A variable or literal string.
string	*envVarValue*	OUT	The value of the environment variable specified in *envVarName*. A null string is returned if the value is not found.

Examples

Example 1: Returns the value for the PATH statement on the OUT parameter

```
@DTW_GETENV(myEnvVarName, myEnvVarValue)
```

- Input: `myEnvVarName = "PATH"`
- Returns: `myEnvVarValue = "/usr/path"`

Example 2: Returns the value for the PATH statement

```
@DTW_rGETENV(myPath)
```

- Input: `myPath = "PATH"`
- Returns: `"/usr/path"`

Example 3: Returns the value for the name of the server

```
The server is @DTW_rGETENV("SERVER_NAME").
```

- Returns: `"www.software.ibm.com"`

DTW_GETINIDATA

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the value of the specified configuration variable. If a value is not found, a null string is returned.

| **Restriction:** For non-OS/400 operating systems, configuration path variables (MACRO_PATH, EXEC_PATH, and INCLUDE_PATH), as well as ENVIRONMENT statements, cannot be retrieved with this call. On the OS/400 operating system, this restriction applies only to ENVIRONMENT statements.

Format

@DTW_GETINIDATA(iniVarName, iniVarValue)

@DTW_rGETINIDATA(iniVarName)

Values

Table 30. DTW_GETINIDATA Parameters

Data Type	Parameter	Use	Description
string	*iniVarName*	IN	A variable or literal string.
string	*iniVarValue*	OUT	The value of the configuration variable specified in *iniVarName.*

Examples

Example 1: Returns the Net.Data path variable value

```
@DTW_GETINIDATA(myEnvVarName, myEnvVarValue)
```

- Input: `myEnvVarName = "FFI_PATH"`
- Returns: `myEnvVarValue = "D:\FFI"`

DTW_HTMLENCODE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Encodes characters using standard HTML decimal escape codes for many, but not all, characters. You can use this function to encode data that you do not want the Web browser to interpret as HTML. For example, by using the appropriate escape character, you can display less-than (<) and greater-than (>) signs, which are usually reserved for HTML tags.

In a second example, the following string in HTML only shows one space between each number:

```
1    2     3
```

Use DTW_HTMLENCODE to ensure that the right number of spaces are shown.

Table 31 shows the characters that are encoded by the DTW_HTMLENCODE function.

Table 31. HTML Decimal Escape Characters

Character	**Name**	**Code**
SPACE	Space	&#32;
"	Double quote	&#34;
#	Number sign	&#35;
%	Percent	&#37;
&	Ampersand	&#38;
[	Left bracket	&#40;
]	Right bracket	&#41;
+	Plus	&#43;
\	Slash	&#47;
:	Colon	&#58;
;	Semicolon	&#59;
<	Less than	&#60;
=	Equals	&#61:
>	Greater than	&#62:
?	Question mark	&#63:
@	At sign	&#64;
/	Backslash	&#92;
ˆ	Carat	&#94;
{	Left brace	&#123;
|	Straight line	&#124;
}	Right brace	&#125;
˜	Tilde	&#126;

Format

@DTW_HTMLENCODE(stringIn, stringOut)

@DTW_rHTMLENCODE(stringIn)

Values

Table 32. DTW_HTMLENCODE Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
string	*stringOut*	OUT	A variable containing the modified input string in which certain characters have been replaced by the encoded HTML escape characters.

Examples

Example 1: Encodes the space character

```
@DTW_HTMLENCODE(string1,string2)
```

- Input: `string1 = "Jim's dog"`
- Returns: `string2 = "Jim's&#32;dog"`

Example 2: Encodes spaces, the less-than sign, and the equal sign

```
@DTW_rHTMLENCODE("X <= 10")
```

- Returns: `"X&#32;&#60;&#61;&#32;10"`

DTW_QHTMLENCODE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

| Performs the same function as @DTW_HTMLENCODE but also encodes the single-quote character (') to &#39;. The HTML decimal escape characters that DTW_QHTMLENCODE uses are shown in Table 31 on page 125.

Format

@DTW_QHTMLENCODE(stringIn, stringOut)

@DTW_rQHTMLENCODE(stringIn)

Values

Table 33. DTW_QHTMLENCODE Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
string	*stringOut*	OUT	A variable that contains the modified form of *stringIn* in which certain characters are replaced by the encoded HTML escape characters.

Examples

Example 1: Encodes an apostrophe and a space

`@DTW_QHTMLENCODE(string1,string2)`

- Input: `string1 = "Jim's dog"`
- Returns: `string2 = "Jim&#39;s&#32;dog"`

Example 2: Encodes apostrophes, spaces, and an ampersand

`@DTW_rQHTMLENCODE("John's & Jane's")`

- Returns: `"John&#39;s&#32;&#38;&#32;Jane&#39;s"`

DTW_SENDMAIL

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X			X	X	X

Purpose

Dynamically builds and transmits electronic mail (e-mail) messages.

This function works with an optional configuration variable, DTW_SMTP_SERVER, which specifies the SMTP server to use for transmitting e-mail messages. The value of this parameter can either be a host name or an IP address. When this variable is not defined, Net.Data uses the local host as the SMTP server. See the configuration chapter in the *Net.Data Administration and Programming Guide* to learn more about these variables.

National Language Issues: Standard Simple Mail Transfer Protocol (SMTP) servers accept only 7-bit data, such as U.S. ASCII characters. If your message has 8-bit characters, it is recommended that you specify an Extended Simple Mail Transfer Protocol (ESMTP) server; ESMTP servers accept 8-bit characters. Net.Data does not encode your 8-bit data into 7-bit data. If you do not have access to an ESMTP server, remove all 8-bit characters from the e-mail message.

Troubleshooting: The following list describes conditions under which Net.Data does not send an e-mail message:

- The specified SMTP server cannot be reached.
- The specified SMTP server does not support the Extended Simple Mail Transfer Protocol (ESMTP), but the specified e-mail message contains non-U.S. ASCII characters.

Format

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject, IN CarbonCopy, IN BlindCarbonCopy, IN ReplyTo, IN Organization)

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject, IN CarbonCopy, IN BlindCarbonCopy, IN ReplyTo

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject, IN CarbonCopy, IN BlindCarbonCopy

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject, IN CarbonCopy

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message

Values

Table 34. DTW_SENDMAIL Parameters

Data Type	Parameter	Use	Description
string	*sender*	IN	A variable or literal string that specifies the author's address. This parameter is required. Valid formats are: • Name <user@domain> • <user@domain> • user@domain

Table 34. DTW_SENDMAIL Parameters (continued)

Data Type	Parameter	Use	Description
string	*recipient*	IN	A variable or literal string that specifies the e-mail addresses to which this message will be sent. This value can contain multiple recipients, separated by a comma (,). This parameter is required. Valid *recipient* formats are: • Name <user@domain> • <user@domain> • user@domain
string	*message*	IN	A variable or literal string that contains the text of the e-mail message. This parameter is required.
string	*subject*	IN	A variable or literal string that contains the text of subject line.
string	*CarbonCopy*	IN	A variable or literal string that contains the e-mail addresses, or names and e-mail addresses of additional recipients. This value can contain multiple additional recipients separated by a comma (,). See the *Recipient* parameter for valid recipient formats.
string	*BlindCarbonCopy*	IN	A variable or literal string that contains the e-mail addresses, or names and e-mail addresses of additional recipients, but the recipients do not appear in the e-mail header. This value can contain multiple additional recipients separated by a comma (,). See the *Recipient* parameter for valid recipient formats.
string	*ReplyTo*	IN	A variable or literal string that contains the e-mail address to which replies to this message should be sent. Valid *ReplyTo* formats are: • Name <user@domain> • <user@domain> • user@domain
string	*Organization*	IN	A variable or literal string that contains the organization name of the *sender*.

Examples

Example 1: Function call that builds and sends a simple e-mail message

```
@DTW_SENDMAIL("<andreb@ibm.com>", "<juliew@ibm.com>", "There is a meeting at 9:30.",
"Status meeting"
```

The DTW_SENDMAIL function sends an e-mail message with the following information:

```
Date: Mon, 3 Apr 1998 09:54:33 PST
To: <juliew@ibm.com>
From: <andreb@ibm.com>
```

```
Subject: Status meeting

There is a meeting at 9:30.
```

The information for `Date` is constructed by using the system date and time functions and is formatted in a SMTP-specific data format.

Example 2: Function call that builds and sends an e-mail message with multiple recipients, carbon copy and blind carbon copy recipients, and the company name

```
@DTW_SENDMAIL("Michael Pauser <michael@ibm.com>", "Andre Beck <abeck@ibm.com>, Julie Wood <juliew@i
```

The DTW_SENDMAIL function sends an e-mail message with the following information:

```
Date: Mon, 3 Apr 1998 09:54:33 PST
To: Andre Beck <abeck@ibm.com>, Julie Wood <juliew@ibm.com>, Debby Nakamura <debbyn@ibm.com>
CC: Dave Hernandez <davehern@ibm.com>
BCC: Anita Chiu <anitac@ibm.com>
From: Michael Pauser <michael@ibm.com>
ReplyTo: meeting@ibm.com
Organzation: IBM
Subject: Status meeting

There is a meeting at 9:30.
```

Example 3: Macro that builds and sends e-mail through a Web form interface

```
%HTML(start) {
<html>
<body>
<h1>Net.Data E-Mail Example</h1>
<form method="post" action="sendemail">
<p>To:<br><input name="recipient"><p>
Subject:<br><input name="subject"><p>
Message:<br><textarea name=message rows=20 cols=40>
</textarea><p>
<input type="submit" value="Send E-mail"><br>
</form>
</body>
</html>
%}

%HTML(sendemail) {
<html>
<body>
<h1>Net.Data E-Mail Example</h1>
@DTW_SENDMAIL("Net.Data E-mail Service <netdata@us.ibm.com>", recipient, message, subject)
<p>E-mail has been sent out.
</body>
</html>
%}
```

This macro sends e-mail through a Web form interface. The HTML start section displays a form into which the recipient's e-mail address, a subject, and a message can be typed. When the user clicks on the **Send E-mail** button, the message is sent out to the recipients specified in the HTML(sendemail) section. This section calls DTW_SENDMAIL and uses the parameters obtained from the Web form to determine the content of the e-mail message, as well as the sender and recipients. Once the e-mail messages have been sent, a confirmation notice is displayed.

Example 4: A macro that uses an SQL query to determine the list of recipients

```
%Function(DTW_SQL) mailing_list(IN message) {
  SELECT EMAIL_ADDRESS FROM CUSTOMERS WHERE ZIPCODE='CA'
    %REPORT {
      Sending out product information to all customers who live in California...<P>
      %ROW {
        @DTW_SENDMAIL("John Doe Corp. <John.Doe@doe.com>", V1, message, "New Product Release")
        E-mail sent out to customer $(V1).<BR>
      %}
    %}
  %}
```

This macro sends out an automated e-mail message to a specified group of customers determined by the results of a SQL query from the customer database. The SQL query also retrieves the e-mail addresses of the customers. The e-mail contents are determined by the value of *message* and can be static or dynamic (for example, you could use another SQL query to dynamically specify the version number of the product or the prices of various offerings).

DTW_SETCOOKIE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X			X	X	X

Purpose

Defines a cookie name, value, and options, such as expiration date and security requirement.

To retrieve a cookie, use the DTW_GETCOOKIE() function. See "DTW_GETCOOKIE" on page 121 to learn how to define a cookie.

When the `secure` requirement is not specified, the cookie can be sent over unsecured channels. The secure option does not require that the browser encrypt the cookie, nor does it ensure that the page containing the DTW_SETCOOKIE statement is transmitted over SSL.

Tips:

- Define and retrieve a cookie in two separate HTTP requests. Because a cookie is visible only after it has been sent to the client, if a macro tries to get a cookie that was defined in the same HTTP request, you might receive unexpected results.
- For simplicity, avoid using semicolons, commas, and spaces as a part of a cookie. When they are required, use the Net.Data function DTW_rURLESCSEQ to process the string that contains the special characters before passing it to DTW_SETCOOKIE. For example,

  ```
  @DTW_SETCOOKIE("my_cookie_name", @DTW_rURLESCSEQ("my cookie value"))
  ```

Restriction:

- If the client Web browser does not support Java Script, the browser does not set the cookie.
- Because DTW_SETCOOKIE generates Java Script code, do not call DTW_SETCOOKIE inside a <SCRIPT> or <NOSCRIPT> HTML element.

Format

@DTW_SETCOOKIE(IN cookie_name, IN cookie_value, IN advanced_options)

@DTW_SETCOOKIE(IN cookie_name, IN cookie_value)

Values

Table 35. DTW_SETCOOKIE Parameters

Data Type	Parameter	Use	Description
string	*cookie_name*	IN	A variable or literal string that specifies the name of the cookie
string	*cookie_value*	IN	A variable or literal string the specifies the value of the cookie.

Table 35. DTW_SETCOOKIE Parameters (continued)

Data Type	Parameter	Use	Description
string	*advanced_options*	IN	A string that contains optional attributes, separated by semicolons, that are used to define the cookie. These attributes are: **expires** = *date* Specifies a date string that defines the valid lifetime of the cookie. After the date expires, the cookie is not longer stored or retrieved. Syntax: `weekday, DD-month-YYYY HH:MM:SS GMT` Where: *weekday* Specifies the full name of the weekday. *DD* Specifies the numerical date of the month. *month* Specifies the three-character abbreviation of the month. *YYYY* Specifies the four-character number of the year. *HH:MM:SS* Specifies the timestamp with hours, minutes, and seconds. **domain** = *domain_name* Specifies the domain attributes of the cookie, for use in domain attribute matching. **path** = *path* Specifies the subset of URLs in a domain for which the cookie is valid. **secure** Specifies that the cookie is transmitted only over secured channels to HTTPS servers.

Examples

Example 1: Defines cookies that contain user ID and password information with the Secure advanced option

```
@DTW_SETCOOKIE("mycookie_name_for_userID", "User1")
@DTW_SETCOOKIE("mycookie_name_for_password", "sd3dT", "secure")
```

Example 2: Defines cookies that contain the expiration date advanced option

```
@DTW_SETCOOKIE("mycookie_name_for_userID", "User1", "expires=Wednesday,
   01-Dec-2010 00:00:00")
@DTW_SETCOOKIE("mycookie_name_for_password", "sd3dT", "expires=Wednesday,
   01-Dec-2010 00:00:00; secure")
```

Example 3: Determines if a cookie for a user exists before gathering user information

```
%HTML(welcome) {
  <html>
  <body>
  <h1>Net.Data Club</h1>
  @DTW_GETCOOKIE("NDC_name", name)
  %IF ($(RETURN_CODE) == "8000") %{ The cookie is not found. %}
  <form method="post" action="remember">
  <p>Welcome to the club. Please enter your name.<br>
  <input name="name">
  <input type="submit" value="submit"><br>
  </form>
  %ELSE
  <p>Hi, $(name). Welcome back.
  %ENDIF
  </body>
  </html>
  %}
```

The HTML(welcome) section checks whether the cookie `NDC_name` exists. If the cookie exists, the browser displays a personalized greeting. If the cookie does not exist, the browser prompts for the user's name, and posts it to the HTML(remember) section. This section records the user's name into the cookie `NDC_name` as shown below:

```
%HTML(remember) {
  <html>
  <body>
  <H1>Net.Data Club>
  @DTW_SETCOOKIE("NDC_name", name, "expires=Wednesday, 01-Dec-2010 00:00:00;path=/")
  <p>Thank you.
  <p><a href="welcome">Come back</a>
  </body>
  </html>
  %}
```

DTW_SETENV

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Assigns an environment variable with a specified value and returns the previous value. A null string is returned if no previous value is found.

Format

@DTW_SETENV(envVarName, envVarValue, prevValue)
@DTW_rSETENV(envVarName, envVarValue)

Values

Table 36. DTW_SETENV Parameters

Data Type	Parameter	Use	Description
string	*envVarName*	IN	A variable or literal string representing the environment variable.
string	*envVarValue*	OUT	A variable or literal string with the value to which the environment variable is assigned.
string	*prevValue*	OUT	A variable that contains the previous value of the environment variable.

Examples

Example 1: Returns the value for the previous path

```
@DTW_SETENV("PATH", "myPath", prevValue)
```

- Input: `myPath = "myPath"`
- Returns: `prevValue = "myPreviousPath"`

Example 2: Returns the value for the previous path and assigns the value for PATH value

```
@DTW_rSETENV("PATH", "myPath")
```

- Input: `myPath = "myPath"`
- Returns: `"myPreviousPath", PATH = "myPath"`

DTW_TIME

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the current system time in the specified format.

Format

@DTW_TIME(stringIn, stringOut)
@DTW_TIME(stringOut)
@DTW_rTIME(stringIn)
@DTW_rTIME()

Values

Table 37. DTW_TIME Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string specifying the time format. Valid formats are: C - Civil time (hh:mmAM/PM using a 12-hour clock) L - Local time (hh:mm:ss) N - Normal time (hh:mm:ss using a 24-hour clock); default H - Number of hours since midnight M - Number of minutes since midnight S - Number of seconds since midnight
string	*stringOut*	OUT	A variable that contains the time in the specified format.

Examples

Example 1: Twenty-four hour clock format

```
@DTW_TIME(results)
```

- Returns: `results = "10:30:53"`

Example 2: Civil time format

```
@DTW_TIME("C", results)
```

- Returns: `results = "10:30AM"`

Example 3: Returns the number of minutes since midnight with the function call

```
@DTW_rTIME("M")
```

- Returns: ″630″

Example 4: Returns the default time and data formats with the function call

```
%REPORT{
<P>This report was created at @DTW_rTIME(), @DTW_rDATE().
%}
```

- Returns: `This report was created 15:04:39, 01 May 1997.`

DTW_URLESCSEQ

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Replaces characters that are not allowed in a URL with their escape values, also known as URL-encoded values. You must use this function to pass any of the characters listed in Table 38 to another macro file or HTML block.

Table 38. Characters Not Allowed in URLs

Character	**Name**	**Code**
SPACE	Space	%20
"	Double quote	%22
#	Number sign	%23
%	Percent	%25
&	Ampersand	%26
+	Plus	%2B
\	Backslash	%2F
:	Colon	%3A
;	Semicolon	%3B
<	Less than	%3C
=	Equals	%3D
>	Greater than	%3E
?	Question mark	%3F
@	At sign	%40
[	Left bracket	%5B
/	Slash	%5C
]	Right bracket	%5D
ˆ	Carat	%5E
{	Left brace	%7B
|	Straight line	%7C
}	Right brace	%7D
˜	Tilde	%7E

Format

@DTW_URLESCSEQ(stringIn, stringOut)

@DTW_rURLESCSEQ(stringIn)

Values

Table 39. DTW_URLESCSEQ Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn*	IN	A variable or literal string.

Table 39. DTW_URLESCSEQ Parameters (continued)

Data Type	Parameter	Use	Description
string	*stringOut*	OUT	A variable containing the input string with characters that are not allowed in URLs that are replaced with their hexadecimal escape values.

Examples

Example 1: Replaces the spaces and an ampersand in *string1* with their URL escape codes and assigns the result to *string2*

```
@DTW_URLESCSEQ(string1,string2)
```

- Input: `string1 = "Guys & Dolls"`
- Returns: `string2 = "Guys%20%26%20Dolls"`

Example 2: Converts spaces and an ampersand to URL-encoded format

```
@DTW_rURLESCSEQ("Guys & Dolls")
```

- Returns: `"Guys%20%26%20Dolls"`

Example 3: Uses DTW_rURLESCSEQ in a ROW block, and converts spaces and the at sign to URL-encoded format

```
%ROW{
<P><a href="fullRpt.mac/input?name=@DTW_rURLESCSEQ(V1)&email=@DTW_rULRESCSEQ(V2)">
$(V1)</a>
%}
```

- Input: V1=″Patrick O’Brien″, V2=″obrien@ibm.com″
- Returns:

  ```
  <P><a href="fullrpt.mac/input?name=Patrick%20O'Brien&email="obrien%40ibm.com">
  Patrick O'Brien</a>
  ```

When the application user clicks on the name, the name and e-mail address are sent to the input block of the Net.Data macro `fullrpt.mac` with the encoded values as variables *name* and *email*.

Math Functions

These functions let you do mathematical calculations.

Performance tip for UNIX, Windows NT, and OS/2: You can optimize the performance of mathematical functions with the DTW_OPTIMIZE_MATH configuration value by setting it to YES in the Net.Data initialization file or the macro file.

- When set to YES, Net.Data uses C mathematical formatting and the functions run faster; however the output format is different than without this variable. Trailing zeros after a decimal point are not displayed.
- When DTW_OPTIMIZE_MATH is set to NO, Net.Data uses REXX mathematical formatting. Functions run slower, but provide output formats that are consistent with the output generated by previous versions of Net.Data. The default value is NO.

See the configuration variables section of the *Net.Data Administration and Programming Guide* to learn how to configure this variable.

NLS considerations for math functions: Net.Data displays decimal points in numerical values based on regional settings specified at the Web server under which Net.Data is running. For example, if the decimal point is specified as a comma (,) at the Web server, Net.Data uses the comma to format decimal data. Net.Data uses the following settings to determine which character is used to specify a decimal point:

For OS/390, Windows NT, OS/2, and UNIX operating systems:
The LOCALE setting at the Web server

For the OS/400 operating system:

- V4R2 or subsequent releases: specified by the user profile under which the process is running.
- V4R1 or previous releases: retrieved from the QDECFMT system value.

The following functions are available for mathematical calculations:

- “DTW_ADD” on page 140
- “DTW_DIVIDE” on page 141
- “DTW_DIVREM” on page 142
- “DTW_FORMAT” on page 143
- “DTW_INTDIV” on page 146
- “DTW_MULTIPLY” on page 147
- “DTW_POWER” on page 148
- “DTW_SUBTRACT” on page 149

DTW_ADD

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Adds the values of two parameters.

Format

@DTW_ADD(number1, number2, precision, result)

@DTW_rADD(number1, number2, precision)

Values

Table 40. DTW_ADD Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number.
float	*number2*	IN	A variable or literal string representing a number.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains the sum of *number1* and *number2*.

Examples

Example 1:

```
@DTW_ADD(NUM1, NUM2, "2", result)
```

- Input: `NUM1 = "105", NUM2 = "3"`
- Returns: `result = "1.1E+2"`

Example 2:

```
@DTW_rADD("12", NUM2, "5")
```

- Input: `NUM2 = "7.00"`
- Returns: `"19.00"`

DTW_DIVIDE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Divides the value of the first parameter by the value of the second parameter.

Format

@DTW_DIVIDE(number1, number2, precision, result)

@DTW_rDIVIDE(number1, number2, precision)

Values

Table 41. DTW_DIVIDE Parameters

Data Type	**Parameter**	**Use**	**Description**
float	*number1*	IN	A variable or literal string representing a number.
float	*number2*	IN	A variable or literal string representing a number.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains the result of *number1* divided by *number2*.

Examples

Example 1:

```
@DTW_DIVIDE("8.0", NUM2, result)
```

- Input: `NUM2 = "2"`
- Returns: `result = "4"`

Example 2:

```
@DTW_rDIVIDE("1", NUM2, "5")
```

- Input: `"1", NUM2 = "3"`
- Returns: `"0.33333"`

Example 3:

```
@DTW_rDIVIDE(NUM1, "2", "5")
```

- Input: `NUM1 = "5"`
- Returns: `"2.5"`

DTW_DIVREM

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

| Divides the first parameter by the second parameter and returns the remainder. The
| sign of the remainder, if nonzero, is the same as that of the first parameter.

Format

@DTW_DIVREM(number1, number2, precision, result)

@DTW_rDIVREM(number1, number2, precision)

Values

Table 42. DTW_DIVREM Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number.
float	*number2*	IN	A variable or literal string representing a number.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains the remainder of *number1* divided by *number2*.

Examples

Example 1:

```
@DTW_DIVREM(NUM1, NUM2, result)
```

- Input: `NUM1 = "2.1", NUM2 = "3"`
- Returns: `result = "2.1"`

Example 2:

```
@DTW_rDIVREM("10", NUM2)
```

- Input: `NUM2 = "0.3"`
- Returns: `"0.1"`

Example 3:

```
@DTW_rDIVREM("3.6", "1.3")
```

- Returns: `"1.0"`

Example 4:

```
@DTW_rDIVREM("-10", "3")
```

- Returns: `"-1"`

DTW_FORMAT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Customizes the formatting for a number. If only the *number* parameter is specified, the result is formatted just as if @DTW_rADD(number,“0”) was performed. If any other options are specified then the number is formatted according to the following rules:

- The *before* and *after* parameters describe how many characters are used for the integer and decimal parts of the *result* parameter, respectively. If you omit either or both of these parameters, the number of characters used for that part is as many as is needed.
- If the *before* parameter is not large enough to contain the integer part of the number (plus the sign for a negative number), an error results. If the *before* parameter is larger than needed for that part, the *number* parameter value is padded on the left with blanks. If the *after* parameter is not the same size as the decimal part of the *number* parameter, the number is rounded (or extended with zeros) to fit. Specifying 0 causes the number to be rounded to an integer.
- In addition, the *expp* and *expt* parameters control the exponent part of the result. The *expp* parameter sets the number of places for the exponent part; the default is to use as many as is needed (which may be zero). The *expt* parameter sets the trigger point for use of exponential notation. The default is the default value of the precision parameter.
- If *expp* is 0, no exponent is supplied and the number is expressed in simple form with added zeros as necessary. If *expp* is not large enough to contain the exponent, an error results.
- If the number of places needed for the integer or decimal part exceeds *expt* or twice *expt*, respectively, use the exponential notation. If *expt* is 0, exponential notation is always used unless the exponent is 0. (If *expp* is 0, this overrides a 0 value of *expt*.) If the exponent is 0 when a nonzero *expp* is specified, then *expp*+2 blanks are supplied for the exponent part of the result. If the exponent is 0 and *expp* is not specified, the simple form is used.

Format

@DTW_FORMAT(number, before, after, expp, expt, precision, result)

@DTW_rFORMAT(number, before, after, expp, expt, precision)

Values

Table 43. DTW_FORMAT Parameters

Data Type	Parameter	Use	Description
float	*number*	IN	A variable or literal string representing a number.
integer	*before*	IN	A variable or literal string representing a positive whole number. This is an optional parameter. You must enter a null string ("") to have additional parameters.
integer	*after*	IN	A variable or literal string representing a positive whole number. This is an optional parameter. You must enter a null string ("") to specify additional parameters.

Table 43. DTW_FORMAT Parameters (continued)

Data Type	Parameter	Use	Description
integer	*expp*	IN	A variable or literal string representing a positive whole number. You must specify a null string ("") to specify additional parameters.
integer	*expt*	IN	A variable or literal string representing a positive whole number. You must enter a null string ("") to specify additional parameters.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains the number with the specified rounding and formatting.

Examples

Example 1:

```
@DTW_FORMAT(NUM, BEFORE, result)
```

- Input: `NUM = "3", BEFORE = "4"`
- Returns: `result= " 3"`

Example 2:

```
@DTW_FORMAT("1.73", "4", "0", result)
```

- Returns: `result = " 2"`

Example 3:

```
@DTW_FORMAT("1.73", "4", "3", result)
```

- Returns: `result = " 1.730"`

Example 4:

```
@DTW_FORMAT(" - 12.73", "", "4", result)
```

- Returns: `result = "-12.7300"`

Example 5:

```
@DTW_FORMAT("12345.73", "", "", "2", "2", result)
```

- Returns: `result = "1.234573E+04"`

Example 6:

```
@DTW_FORMAT("1.234573", "", "3", "", "0", result)
```

- Returns: `result = "1.235"`

Example 7:

```
@DTW_rFORMAT(" - 12.73")
```

- Returns: `" – 12.73"`

Example 8:

```
@DTW_rFORMAT("0.000")
```

- Returns: `"0"`

Example 9:

```
@DTW_rFORMAT("12345.73", "", "", "3", "6")
```

- Returns: `"12345.73"`

Example 10:

```
@DTW_rFORMAT("1234567e5", "", "3", "0")
```

- Returns: ″123456700000.000″

Example 11:

```
@DTW_rFORMAT("12345.73", "", "3", "", "0")
```

- Returns: `"1.235E+4"`

DTW_INTDIV

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Divides the first parameter by the second parameter and returns the integer part of the result.

Format

@DTW_INTDIV(number1, number2, precision, result)

@DTW_rINTDIV(number1, number2, precision)

Values

Table 44. DTW_INTDIV Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number.
float	*number2*	IN	A variable or literal string representing a number.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains integer part of *number1* divided by *number2*.

Examples

Example 1:

```
@DTW_INTDIV(NUM1, NUM2, result)
```

- Input: `NUM1 = "10", NUM2 = "3"`
- Returns: `result = "3"`

Example 2:

```
@DTW_rINTDIV("2", NUM2)
```

- Input: `NUM2 = "3"`
- Returns: `"0"`

DTW_MULTIPLY

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Multiplies two parameters and returns the result.

Format

@DTW_MULTIPLY(number1, number2, precision, result)
@DTW_rMULTIPLY(number1, number2, precision)

Values

Table 45. DTW_MULTIPLY Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number.
float	*number2*	IN	A variable or literal string representing a number.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains the product of *number1* and *number2*.

Examples

Example 1:

```
@DTW_MULTIPLY(NUM1, NUM2, result)
```

- Input: `NUM1 = "4", NUM2 = "5"`
- Returns: `result = "20"`

Example 2:

```
@DTW_rMULTIPLY("0.9", NUM2)
```

- Input: `NUM2 = "0.8"`
- Returns: `"0.72"`

DTW_POWER

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Raises the first parameter to the power of the second parameter and returns the result.

Format

@DTW_POWER(number1, number2, precision, result)
@DTW_rPOWER(number1, number2, precision)

Values

Table 46. DTW_POWER Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number.
float	*number2*	IN	A variable or literal string representing a number.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains the result of *number1* raised to the power of *number2*.

Examples

Example 1:

```
@DTW_POWER(NUM1, NUM2, result)
```

- Input: `NUM1 = "2", NUM2 = "-3"`
- Returns: `result = "0.125"`

Example 2:

```
@DTW_rPOWER("1.7", NUM2, precision)
```

- Input: `NUM2 = "8", precision = "5"`
- Returns: `"69.758"`

DTW_SUBTRACT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Subtracts the value of the second parameter from the value of the first parameter and returns the result.

Format

@DTW_SUBTRACT(number1, number2, precision, result)
@DTW_rSUBTRACT(number1, number2, precision)

Values

Table 47. DTW_SUBTRACT Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number.
float	*number2*	IN	A variable or literal string representing a number.
integer	*precision*	IN	A variable or literal string representing a positive whole number that specifies the precision of the result. The default is 9.
float	*result*	OUT	A variable that contains the difference of *number1* and *number2*.

Examples

Example 1:

```
@DTW_SUBTRACT(NUM1, NUM2, comp)
%IF(comp > "0")
<P>$(NUM1) is larger than  $(NUM2).
%ENDIF
```

- Input: `NUM2 = "2.07"`
- Returns: `"-0.77"`

This example shows a way to compare numeric values, which are strings in Net.Data.

Example 2:

```
@DTW_SUBTRACT(NUM1, NUM2, result)
```

- Input: NUM1 = `"1.3, NUM2 = "1.07"`
- Returns: result = `"0.23"`

Example 3:

```
@DTW_rSUBTRACT("1.3", NUM2)
```

- Input: `NUM2 = "2.07"`
- Returns: `"-0.77"`

String Functions

The following functions are the set of standard string functions that Net.Data supports:

- "DTW_ASSIGN" on page 151
- "DTW_CONCAT" on page 152
- "DTW_DELSTR" on page 153
- "DTW_INSERT" on page 154
- "DTW_LASTPOS" on page 156
- "DTW_LENGTH" on page 157
- "DTW_LOWERCASE" on page 158
- "DTW_POS" on page 159
- "DTW_REVERSE" on page 160
- "DTW_STRIP" on page 161
- "DTW_SUBSTR" on page 162
- "DTW_TRANSLATE" on page 163
- "DTW_UPPERCASE" on page 165

MBCS support for OS/390, OS/2, Windows NT, and UNIX: You can specify multiple-byte character set (MBCS) support for word and string functions with the DTW_MBMODE configuration value. Specify this value in the Net.Data initialization file; the default is no support. You can override the value in the initialization file by setting the DTW_MBMODE variable in a Net.Data macro file. See the configuration variable section in *Net.Data Administration and Programming Guide* and "DTW_MBMODE" on page 89 for more information.

MBCS support for OS/400: DBCS support is provided automatically and does not require this variable.

DTW_ASSIGN

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Assigns the value of an input variable to an output variable. Because $(V*n*), where *n* is a number, is not recognized outside the ROW block, you can use this function to assign the value to a different variable if you want to reference the value outside the ROW block.

You can also use this function to change a variable in a macro. For example, you can change DATABASE for an HTML block. (See “DATABASE” on page 82 for an example.)

Format

@DTW_ASSIGN(stringOut, stringIn)

Values

Table 48. DTW_ASSIGN Parameters

Data Type	Parameter	Use	Description
string	*stringOut*	OUT	A variable that contains the literal string identical to *stringIn*.
string	*stringIn*	IN	A variable or literal string.

Examples

Example 1:

```
@DTW_ASSIGN(RC, "0")
```

- Sets RC to ″0″.

Example 2:

```
@DTW_ASSIGN(string1, string2)
```

- Sets *string1* to the value of *string2*.

DTW_CONCAT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Concatenates two strings.

Format

@DTW_CONCAT(stringIn1, stringIn2, stringOut)

@DTW_rCONCAT(stringIn1, stringIn2)

Values

Table 49. DTW_CONCAT Parameters

Data Type	Parameter	Use	Description
string	*stringIn1*	IN	A variable or literal string.
string	*stringIn2*	IN	A variable or literal string.
string	*stringOut*	OUT	A variable that contains the string `'stringIn1stringIn2'`, where *string1* is concatenated with *string2*.

Examples

Example 1:

```
@DTW_CONCAT("This", " is a test.", result)
```

- Returns: `result = "This is a test."`

Example 2:

```
@DTW_CONCAT(string1, "1-2-3", result)
```

- Input: `string1 = "Testing "`
- Returns: `result = "Testing 1-2-3"`

Example 3:

```
@DTW_rCONCAT("This", " is a test.")
```

- Returns: `"This is a test."`

DTW_DELSTR

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Deletes a substring of the specified string from the *n*th character for length characters.

Format

@DTW_DELSTR(stringIn, n, length, stringOut)
@DTW_DELSTR(stringIn, n, stringOut)
@DTW_rDELSTR(stringIn, n, length)
@DTW_rDELSTR(stringIn, n)

Values

Table 50. DTW_DELSTR Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn*	IN	A variable or literal string.
integer	*n*	IN	The position of the character at which the substring to delete begins. If *n* is greater than the length of *stringIn*, *stringOut* is set to the value of *stringIn*.
integer	*length*	OUT	The length of the substring to delete. The default is to delete all characters to the end of *stringIn*.
string	*stringOut*	OUT	A variable that contains the modified form of *stringIn*.

Examples

Example 1:

```
@DTW_DELSTR("abcde", "3", "2", result)
```

- Returns: `result = "abe"`

Example 2:

```
@DTW_rDELSTR("abcde", "4", "1")
```

- Returns: `"abce"`

DTW_INSERT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Inserts a string into another string starting after the *n*th character.

Format

@DTW_INSERT(stringIn1, stringIn2, n, length, pad, stringOut)
@DTW_INSERT(stringIn1, stringIn2, n, length, stringOut)
@DTW_INSERT(stringIn1, stringIn2, n, stringOut)
@DTW_INSERT(stringIn1, stringIn2, stringOut)
@DTW_rINSERT(stringIn1, stringIn2, n, length, pad)
@DTW_rINSERT(stringIn1, stringIn2, n, length)
@DTW_rINSERT(stringIn1, stringIn2, n)
@DTW_rINSERT(stringIn1, stringIn2)

Values

Table 51. DTW_INSERT Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn1*	IN	A variable or literal string to be inserted into *stringIn2*.
string	*stringIn2*	IN	A variable or literal string.
integer	*n*	IN	The character position in *stringIn2* after which *stringIn1* is inserted. If *n* is greater than the length of *stringIn2*, it is padded with the padding character, *pad*, until it has enough characters. The default is to insert at the beginning of *stringIn2*.
integer	*length*	IN	The number of characters of *stringIn1* to insert. The string is padded with the padding character, *pad*, if this parameter is greater than the length of *stringIn1*. The default is the length of *stringIn1*.
integer	*pad*	IN	The padding character, as described for *n* and *length*. The default pad character is a blank.
string	*stringOut*	OUT	A variable that contains *stringIn2* modified by inserting part or all of *stringIn1*.

Examples

Example 1:

```
@DTW_INSERT("123", "abc", result)
```

- Returns: `result = "123abc"`

Example 2:

```
@DTW_INSERT("123", "abc", "5", result)
```

- Returns: `result = "abc 123"`

Example 3:

```
@DTW_INSERT("123", "abc", "5", "6", result)
```

- Returns: `result = "abc 123 "`

Example 4:

```
@DTW_INSERT("123", "abc", "5", "6", "/", result)
```

- Returns: result = `"abc//123///"`

Example 5:

```
@DTW_rINSERT("123", "abc", "5", "6", "+")
```

- Returns: `"abc++123+++"`

DTW_LASTPOS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the position of the last occurrence of a string in another string, starting from the *n*th character and working backwards (right to left).

Format

@DTW_LASTPOS(stringIn1, stringIn2, n, position)

@DTW_LASTPOS(stringIn1, stringIn2, position)

@DTW_rLASTPOS(stringIn1, stringIn2, n)

@DTW_rLASTPOS(stringIn1, stringIn2)

Values

Table 52. DTW_LASTPOS Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn1*	IN	A variable or literal string searched for in *stringIn2*.
string	*stringIn2*	IN	A variable or literal string.
integer	*n*	IN	The character position in *stringIn2* to begin searching for *stringIn1*. The default is to start searching at the last character and scan backwards (from right to left).
integer	*position*	OUT	The position of the last occurrence of *stringIn1* in *stringIn2*. If no occurrence is found, 0 is returned.

Examples

Example 1:

```
@DTW_LASTPOS(" ", "abc def ghi", result)
```

- Returns: `result = "8"`

Example 2:

```
@DTW_LASTPOS(" ", "abc def ghi", "10", result)
```

- Returns: `result = "8"`

Example 3:

```
@DTW_rLASTPOS(" ", "abc def ghi", "7")
```

- Returns: `"4"`

DTW_LENGTH

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the length of a string.

Format

@DTW_LENGTH(stringIn, length)

@DTW_rLENGTH(stringIn)

Values

Table 53. DTW_LENGTH Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
integer	*length*	OUT	A symbol containing the number of characters in *stringIn.*

Examples

Example 1:

```
@DTW_LENGTH("abcdefgh", result)
```

- Returns: `result = "8"`

Example 2:

```
@DTW_rLENGTH("")
```

- Returns: `"0"`

DTW_LOWERCASE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns a string in all lowercase.

Format

@DTW_LOWERCASE(stringIn, stringOut)
@DTW_rLOWERCASE(stringIn)
@DTW_mLOWERCASE(stringMult1, stringMult2, ..., stringMultn)

Values

Table 54. DTW_LOWERCASE Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string with characters of any case.
string	*stringOut*	OUT	A variable that contains *stringIn* with all characters in lowercase.
string	*stringMult*	INOUT	• On input: A variable that contains a string. • On output: A variable that contains the input string converted to lowercase.

Examples

Example 1:

```
@DTW_LOWERCASE("This", stringOut)
```

- Returns: `stringOut = "this"`

Example 2:

```
@DTW_rLOWERCASE(string1)
```

- Input: `string1 = "Hello"`
- Returns: `"hello"`

Example 3:

```
@DTW_mLOWERCASE(string1, string2, string3)
```

- Input: `string1 = "THIS", string2 = "IS", string3 = "LOWERCASE"`
- Returns: `string1 = "this", string2 = "is", string3 = "lowercase"`

DTW_POS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the position of the first occurrence of a string in another string, using a forward search pattern.

Format

@DTW_POS(stringIn1, stringIn2, n, nOut)
@DTW_POS(stringIn1, stringIn2, nOut)
@DTW_rPOS(stringIn1, stringIn2, n)
@DTW_rPOS(stringIn1, stringIn2)

Values

Table 55. DTW_POS Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn1*	IN	A variable or literal string to search for.
string	*stringIn2*	IN	A variable or literal string to search.
integer	*n*	IN	The character position in *stringIn2* to begin searching. The default value is to start searching at the first character of *stringIn2*.
integer	*nOut*	OUT	A variable that contains the position of the first occurrence of *stringIn1* in *stringIn2*. If no occurrence is found, 0 is returned.

Examples

Example 1:

```
@DTW_POS("day", "Saturday", result)
```

- Returns: `result = "6"`

Example 2:

```
@DTW_POS("a", "Saturday", "3", result)
```

- Returns: `result = "7"`

Example 3:

```
@DTW_rPOS(" ", "abc def ghi", "5")
```

- Returns: `"8"`

DTW_REVERSE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Reverses the input string.

Format

@DTW_REVERSE(stringIn, stringOut)

@DTW_rREVERSE(stringIn)

Values

Table 56. DTW_REVERSE Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string to reverse.
string	*stringOut*	OUT	A variable that contains the reversed form of *stringIn*.

Examples

Example 1:

```
@DTW_REVERSE("This is it.", result)
```

- Returns: `result = ".ti si sihT"`

Example 2:

```
@DTW_rREVERSE(string1)
```

- Input: `string1 = "reversed"`
- Returns: `"desrever"`

DTW_STRIP

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Removes leading blanks, trailing blanks, or both from the input string.

Format

@DTW_STRIP(stringIn, option, stringOut)
@DTW_STRIP(stringIn, stringOut)
@DTW_rSTRIP(stringIn, option)
@DTW_rSTRIP(stringIn)

Values

Table 57. DTW_STRIP Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
string	*option*	IN	Specifies which blanks to remove from *stringIn*. The default is B. B or b - remove both leading and trailing blanks L or l - remove leading blanks only T or t - remove trailing blanks only
string	*stringOut*	OUT	A variable that contains *stringIn* with blanks removed as specified by option.

Examples

Example 1:

```
@DTW_STRIP("  day ", result)
```

- Returns: `result = "day"`

Example 2:

```
@DTW_STRIP("  day ", "T", result)
```

- Returns: `result = " day"`

Example 3:

```
@DTW_rSTRIP("  a day  ", "L")
```

- Returns: `"a day "`

DTW_SUBSTR

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns a substring of the input string, with optional pad characters.

Format

@DTW_SUBSTR(stringIn, n, length, pad, stringOut)
@DTW_SUBSTR(stringIn, n, length, stringOut)
@DTW_SUBSTR(stringIn, n, stringOut)
@DTW_rSUBSTR(stringIn, n, length, pad)
@DTW_rSUBSTR(stringIn, n, length)
@DTW_rSUBSTR(stringIn, n)

Values

Table 58. DTW_SUBSTR Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string to be searched.
integer	*n*	IN	The first character position of the substring. The default is to start at the beginning of *stringIn*
integer	*length*	IN	The number of characters of the substring. The default is the rest of the string.
string	*pad*	IN	The padding character used if *n* is greater than the length of *stringIn* or if *length* is longer than *stringIn*. The default is a blank.
string	*stringOut*	OUT	A variable that contains a substring of *stringIn*.

Examples

Example 1:

```
@DTW_SUBSTR("abc", "2", result)
```

- Returns: `result = "bc"`

Example 2:

```
@DTW_SUBSTR("abc", "2", "4", result)
```

- Returns: `result = "bc  "`

Example 3:

```
@DTW_SUBSTR("abc", "2", "4", ".", result )
```

- Returns: `result = "bc.."`

Example 4:

```
@DTW_rSUBSTR("abc", "2", "6", ".")
```

- Returns: `"bc...."`

DTW_TRANSLATE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Translates characters in the input string using input and output translation tables, *tableI* and *tableO*. If no *tableI*, *tableO*, and the *default* character are in the parameter list, the *stringIn* parameter is translated to uppercase. If *tableI* and *tableO* are in the list, each character in the input string is searched for in *tableI* and translated to the corresponding character in *tableO*. If a character in *tableI* has no corresponding character in *tableO*, the *default* character is used instead.

Format

@DTW_TRANSLATE(stringIn, tableO, tableI, default, stringOut)
@DTW_TRANSLATE(stringIn, tableO, tableI, stringOut)
@DTW_TRANSLATE(stringIn, tableO, stringOut)
@DTW_TRANSLATE(stringIn, stringOut)
@DTW_rTRANSLATE(stringIn, tableO, tableI, default)
@DTW_rTRANSLATE(stringIn, tableO, tableI)
@DTW_rTRANSLATE(stringIn, tableO)
@DTW_rTRANSLATE(stringIn)

Values

Table 59. DTW_TRANSLATE Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
string	*tableO*	IN	A variable or literal string used as a translation table. Use null (″″) to specify *tableI* or *default*; otherwise this parameter is optional.
string	*tableI*	IN	A variable or literal string searched for in *stringIn*. Use null (″″) to specify *default*; otherwise this parameter is optional.
string	*default*	IN	The default character to use. The default is a blank.
string	*stringOut*	OUT	A variable that contains the translated result of *stringIn*.

Examples

Example 1:

```
@DTW_TRANSLATE("abbc", result)
```

- Returns: `result = "ABBC"`

Example 2:

```
@DTW_TRANSLATE("abbc", "R", "bc", result)
```

- Returns: `result = "aRR "`

Example 3:

```
@DTW_rTRANSLATE("abcdef", "12", "abcd", ".")
```

- Returns: `"12..ef"`

Example 4:

`@DTW_rTRANSLATE("abbc", "", "", "")`

- Returns: `"abbc"`

DTW_UPPERCASE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns a string in uppercase.

Format

@DTW_UPPERCASE(stringIn, stringOut)
@DTW_rUPPERCASE(stringIn)
@DTW_mUPPERCASE(stringMult1, stringMult2, ..., stringMultn)

Values

Table 60. DTW_UPPERCASE Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn*	IN	A variable or literal string with characters of any case.
string	*stringOut*	OUT	A variable that contains *stringIn* with all characters in uppercase.
string	*stringMult*	INOUT	• On input: A variable that contains a string. • On output: A variable that contains the input string converted to uppercase.

Examples

Example 1:

```
@DTW_UPPERCASE("Test", result)
```

- Returns: `result = "TEST"`

Example 2:

```
@DTW_rUPPERCASE(string1)
```

- Input: `string1 = "Web pages"`
- Returns: `"WEB PAGES"`

Example 3:

```
@DTW_mUPPERCASE(string1, string2, string3)
```

- Input: `string1 = "This", string2 = "is", string3 = "uppercase"`
- Returns: `string1 = "THIS", string2 = "IS", string3 = "UPPERCASE"`

Word Functions

These functions supplement the string functions by modifying words or sets of words. Net.Data interprets a word as a space-delimited string, or a string with spaces on both sides. Here are some examples:

String value	Number of words
one two three	3
one , two , three	5
Part 2: Internet Sales Grow	5

| **MBCS support for OS/390, OS/2, Windows NT, and UNIX:** You can specify multiple-byte character set (MBCS) support for word and string functions with the DTW_MBMODE configuration value. Specify this value in the Net.Data initialization file; the default is no support. You can override the value in the initialization file by setting the DTW_MBMODE variable in a Net.Data macro file. See the configuration variable section in *Net.Data Administration and Programming Guide* and "DTW_MBMODE" on page 89 for more information.

| **MBCS support for OS/400:** DBCS support is provided automatically and does not require this variable.

The following functions are word functions that Net.Data supports:

- "DTW_DELWORD" on page 167
- "DTW_SUBWORD" on page 168
- "DTW_WORD" on page 169
- "DTW_WORDINDEX" on page 170
- "DTW_WORDLENGTH" on page 171
- "DTW_WORDPOS" on page 172
- "DTW_WORDS" on page 173

DTW_DELWORD

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns a substring of the input string. Words are deleted from word *n* for the number of words specified by length.

Format

@DTW_DELWORD(stringIn, n, length, stringOut)
@DTW_DELWORD(stringIn, n, stringOut)
@DTW_rDELWORD(stringIn, n, length)
@DTW_rDELWORD(stringIn, n)

Values

Table 61. DTW_DELWORD Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn*	IN	A variable or literal string.
integer	*n*	IN	The word position of the first word to be deleted.
integer	*length*	IN	The number of words to delete. The default is to delete all words from *n* to the end of *stringIn*. Optional parameter.
string	*stringOut*	OUT	A variable that contains the modified form of *stringIn*.

Examples

Example 1:

```
@DTW_DELWORD("Now is the time", "5", result)
```

- Returns: `result = "Now is the time"`

Example 2:

```
@DTW_DELWORD("Now is the time", "2", result)
```

- Returns: `result = "Now"`

Example 3:

```
@DTW_DELWORD("Now is the time", "2", "2", result)
```

- Returns: `result = "Now time"`

Example 4:

```
@DTW_rDELWORD("Now is the time.", "3")
```

- Returns: `"Now is"`

DTW_SUBWORD

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns a substring of the input string. The substring begins at word *n* and continues for the number of words specified by *length*.

Format

@DTW_SUBWORD(stringIn, n, length, stringOut)
@DTW_SUBWORD(stringIn, n, stringOut)
@DTW_rSUBWORD(stringIn, n, length)
@DTW_rSUBWORD(stringIn, n)

Values

Table 62. DTW_SUBWORD Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
integer	*n*	IN	The word position of the first word of the substring. A null is returned if this value is greater than the number of words in *stringIn*.
integer	*length*	IN	The number of words in the substring. If this value is greater than the number of words from *n* to the end of *stringIn*, all words to the end of *stringIn* are returned. The default is to return all words from *n* to the end of *stringIn*.
string	*stringOut*	OUT	A variable that contains a substring of *stringIn* specified by *n* and *length*.

Examples

Example 1:

```
@DTW_SUBWORD("Now is the   time", "5", result)
```

- Returns: `result = ""`

Example 2:

```
@DTW_SUBWORD("Now is the   time", "2", result)
```

- Returns: `result = "is the time"`

Example 3:

```
@DTW_SUBWORD(Now is the   time", "2", "2", result)
```

- Returns: `result = "is the"`

Example 4:

```
@DTW_rSUBWORD("Now is the   time", "3")
```

- Returns: `"the time"`

DTW_WORD

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns a single word from a specified position of the input string.

Format

@DTW_WORD(stringIn, n, stringOut)
@DTW_rWORD(stringIn, n)

Values

Table 63. DTW_WORD Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
integer	*n*	IN	The word position of the word to return. If this value is greater than the number of words in *stringIn*, a null is returned.
string	*stringOut*	OUT	A variable that contains the word at word position *n*.

Examples

Example 1:

```
@DTW_WORD("Now is the time", "3", result)
```

- Returns: result = ″the″

Example 2:

```
@DTW_WORD("Now is the time", "5", result)
```

- Returns: `result = ""`

Example 3:

```
@DTW_rWORD("Now is the time", "4")
```

- Returns: `"time"`

DTW_WORDINDEX

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the character position of the first character in the *n*th word of the input string.

Format

@DTW_WORDINDEX(stringIn, n, stringOut)

@DTW_rWORDINDEX(stringIn, n)

Values

Table 64. DTW_WORDINDEX Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn*	IN	A variable or literal string.
integer	*n*	IN	The word position of the word to index. If this value is greater than the number of words in the input string, 0 is returned.
string	*stringOut*	OUT	A variable that contains the character position of the *n*th word of *stringIn*.

Examples

Example 1:

```
@DTW_WORDINDEX("Now is the time", "3", result)
```

- Returns: `result = "8"`

Example 2:

```
@DTW_WORDINDEX("Now is the time", "6", result)
```

- Returns: `result = "0"`

Example 3:

```
@DTW_rWORDINDEX("Now is the time", "2")
```

- Returns: ″5″

DTW_WORDLENGTH

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the length of the *n*th word of the input string.

Format

@DTW_WORDLENGTH(stringIn, n, stringOut)
@DTW_rWORDLENGTH(stringIn, n)

Values

Table 65. DTW_WORDLENGTH Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
integer	*n*	IN	The word position of the word whose length you want to know. If this value is greater than the number of words in the input string, 0 is returned.
string	*stringOut*	OUT	A variable that contains the length of the *n*th word in *stringIn*.

Examples

Example 1:

```
@DTW_WORDLENGTH("Now is the time", "1", result)
```

- Returns: `result = "3"`

Example 2:

```
@DTW_rWORDLENGTH("Now is the time", "6")
```

- Returns: `"0"`

DTW_WORDPOS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the word number of the first occurrence of one string within another. Multiple blanks are treated as single blanks for comparison. The comparison is case sensitive.

Format

@DTW_WORDPOS(stringIn1, stringIn2, n, stringOut)
@DTW_WORDPOS(stringIn1, stringIn2, stringOut)
@DTW_rWORDPOS(stringIn1, stringIn2, n)
@DTW_rWORDPOS(stringIn1, stringIn2)

Values

Table 66. DTW_WORDPOS Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn1*	IN	A variable or literal string.
string	*stringIn2*	IN	A variable or literal string to search.
integer	*n*	IN	The word position in *stringIn2* to begin searching. If this value is larger than the number of words in *stringIn2*, 0 is returned. The default is to search from the beginning of *stringIn2*.
string	*stringOut*	OUT	The word position of *stringIn1* in *stringIn2*.

Examples

Example 1:

```
@DTW_WORDPOS("the", "Now is the time", result)
```

- Returns: `result = "3"`

Example 2:

```
@DTW_WORDPOS("The", "Now is the time", result)
```

- Returns: `result = "0"`

Example 3:

```
@DTW_WORDPOS("The", "Now is the time", "5", result)
```

- Returns: `result = "0"`

Example 4:

```
@DTW_WORDPOS("is  the", "Now is    the time", result)
```

- Returns: `result = "2"`

Example 5:

```
@DTW_rWORDPOS("be", "To be or not to be", "3")
```

- Returns: `"6"`

DTW_WORDS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the number of words in a string.

Format

@DTW_WORDS(stringIn, stringOut)
@DTW_rWORDS(stringIn)

Values

Table 67. DTW_WORDS Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*stringIn*	IN	A variable or literal string.
string	*stringOut*	OUT	A variable that contains the number of words in *stringIn*.

Examples

Example 1:

```
@DTW_WORDS("Now is the time", result)
```

- Returns:
  result = ″4″

Example 2:

```
@DTW_rWORDS(" ")
```

- Returns: `"0"`

Table Functions

These functions simplify working with Net.Data tables and are more efficient than writing your own functions using REXX, C, or Perl.

DTW_TB_COLS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Returns the current number of columns in a table.

Format

@DTW_TB_COLS(table, cols)

@DTW_TB_rCOLS(table)

Values

Table 68. DTW_TB_COLS Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable for which the number of columns are returned.
integer	*cols*	OUT	A variable that contains the number of columns in *table*.

Examples

Example 1: Retrieves the number of columns and assigns the value to *cols*

```
%DEFINE myTable = %TABLE
%DEFINE cols = ""
...
@FillTable()
...
@DTW_TB_COLS(myTable, cols)
```

Example 2: Retrieves and displays the value for the current number of columns in the table

```
%DEFINE myTable = %TABLE
...
@FillTable()
...
<P>My table contains @DTW_TB_rCOLS(myTable) columns.
```

DTW_TB_DLIST

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns an HTML definition list from a macro table.

Format

@DTW_TB_DLIST(table, term, def, termstyle, defstyle, link, link_u, image, image_u)

Values

Table 69. DTW_TB_DLIST Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A symbol specifying the macro table variable to display as an HTML definition list.
integer	*term*	IN	The column number in *table* that contains *term* name values (the text to go after the `<DT>` tag). The default is to use the first column.
integer	*def*	IN	The column number in *table* containing term definition values (the text to go after the `<DD>` tag). The default is to use the second column.
string	*termstyle*	IN	A variable or literal string that contains a list of HTML elements for the *term* name values. The default is to use no style tags.
string	*defstyle*	IN	A variable or literal string containing a list of HTML elements for the *term* definition values. The default is to use no style tags.
string	*link*	IN	Specifies for which HTML elements an HTML link is generated. Valid values are DT and DD. The default is not to generate HTML links.
integer	*link_u*	IN	The column number in *table* that contains the URLs for the HTML references. The default is not to generate HTML links.
string	*image*	IN	Specifies for which HTML elements an inline image is generated. Valid values are DT and DD. The default is not to generate inline images (DT).
integer	*image_u*	IN	The column number in *table* that contains the URLs for the inline images. The default is not to generate inline images.

Examples

Example 1: Creates a definition list producing the HTML shown below, depending on the table data

```
@DTW_TB_DLIST(Mytable,"3","4","b i","strong","DD","2","DT","1")
```

Results:

```
<DL>
<DT>
<IMG SRC="http://www.mycompany.com/images/image1.gif" ALT=""><b><i>image1text</i></b>
<DD>
<A HREF="http://www.mycompany.com/link1.html"><strong>link1text</strong></A>
<DT>
<IMG SRC="http://www.mycompany.com/images/image2.gif" ALT=""><b><i>image2text</i></b>
<DD>
<A HREF="http://www.mycompany.com/link2.html"><strong>link2text</strong></A>
<DT>
<IMG SRC="http://www.mycompany.com/images/image3.gif" ALT=""><b><i>image3text</i></b>
<DD>
<A HREF="http://www.mycompany.com/link3.html"><strong>link3text</strong></A>
<DT>
<IMG SRC="http://www.mycompany.com/images/image4.gif" ALT=""><b><i>image4text</i></b>
<DD>
<A HREF="http://www.mycompany.com/link4.html"><strong>link4text</strong></A>
</DT>
</DL>
```

DTW_TB_DUMPH

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the contents of a macro table variable. Each row of the table is displayed on a different line. The entire table is enclosed in <PRE></PRE> tags.

Format

@DTW_TB_DUMPH(table)

Values

Table 70. DTW_TB_DUMPH Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A symbol specifying the macro table variable to display.

Examples

Example 1:

```
@DTW_TB_DUMPH(Mytable)
```

The HTML generated by this example looks like this:

```
<PRE>
Name             Department                Position
Jack Smith       Internet Technologies     Software Engineer
Helen Williams   Database                  Development Manager
Alex Jones       Manufacturing             Industrial Engineer
Tom Baker        Procurement               Sales Rep
</PRE>
```

DTW_TB_DUMPV

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the contents of a macro table variable. Each table value is on a different line. The entire table is enclosed in <PRE></PRE> tags.

Format

@DTW_TB_DUMPV(table)

Values

Table 71. DTW_TB_DUMPV Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A symbol specifying the macro table variable to display.

Examples

Example 1:

```
@DTW_TB_DUMPV(Mytable)
```

The HTML generated for this example looks like this:

```
<PRE>
http://www.mycompany.com/images/image1.gif
http://www.mycompany.com/link1.html
image1text
link1text
http://www.mycompany.com/images/image2.gif
http://www.mycompany.com/link2.html
image2text
link2text
http://www.mycompany.com/images/image3.gif
http://www.mycompany.com/link3.html
image3text
link3text
http://www.mycompany.com/images/image4.gif
http://www.mycompany.com/link4.html
image4text
link4text
</PRE>
```

DTW_TB_GETN

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Retrieves the column heading for the column number specified in *col*.

You must set the number of columns in the table before calling DTW_TB_GETN(). You can set the number of columns with the DTW_TB_SETCOLS() or DTW_TB_INSERTCOL() functions, or by passing the table to a language environment to be set.

Format

@DTW_TB_GETN(table, name, col)

@DTW_TB_rGETN(table, col)

Values

Table 72. DTW_TB_GETN Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable from which a column name is returned.
table	*table*	OUT	A variable that contains the name of the column specified in *col*.
integer	*cols*	IN	The column number of the column whose name is to be returned.

Examples

Example 1: Retrieves the column name of column 4

```
%DEFINE myTable = %TABLE
%DEFINE name = ""
...
@FillTable()
...
@DTW_TB_GETN(myTable, name, "4")
```

Example 2: Retrieves the column name of the last column in the table

```
%DEFINE myTable = %TABLE
...
@FillTable()
...
<P>The column name of the last column is @DTW_TB_rGETN(myTable, @DTW_TB_rCOLS(myTable))
```

DTW_TB_GETV

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X			X	X	X

Purpose

Retrieves the table value for the row and column numbers specified in *row* and *col*.

You must set the number of columns in the table before calling DTW_TB_GETV(). You can set the number of columns with the DTW_TB_SETCOLS() or DTW_TB_INSERTCOL() functions, or by passing the table to a language environment to be set.

Format

@DTW_TB_GETV(table, value, row, col)

@DTW_TB_rGETV(table, row, col)

Values

Table 73. DTW_TB_GETV Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable for which a table value is returned.
integer	*value*	OUT	A variable that contains the value at the row and column specified in *row* and *col*.
integer	*row*	IN	The row number of the value to be returned.
integer	*col*	IN	The column number of the value to be returned.

Examples

Example 1: Retrieves the table value at row 6, column 3

```
%DEFINE myTable = %TABLE
%DEFINE value = ""
...
@FillTable()
...
@DTW_TB_GETV(myTable, value, "6", "3")
```

Example 2: Retrieves the table value at row 1, column 1

```
%DEFINE myTable = %TABLE
...
@FillTable()
...
<P>The table value of row 1, column 1 is @DTW_TB_rGETV(myTable, "1", "1").
```

DTW_TB_HTMLENCODE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns the input macro table with these HTML characters encoded:

Name	Character	Code
Ampersand	&	&#38;
Double quote	″	&#34;
Greater than	>	&#62;
Less than	<	&#60;

Format

@DTW_TB_HTMLENCODE(table, collist)

Values

Table 74. DTW_TB_HTMLENCODE Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable to modify.
string	*collist*	IN	The column numbers in *table* to encode. The default is to encode all columns.

Examples

Example 1:

```
@DTW_TB_HTMLENCODE(Mytable, "3 4")
```

The special characters in columns 3 and 4 of the specified table are replaced with their encoded forms.

DTW_TB_INPUT_CHECKBOX

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns one or more HTML check box input tags from a macro table variable.

Format

@DTW_TB_INPUT_CHECKBOX(table, prompt, namecol, valuecol, rows, checkedrows)

Values

Table 75. DTW_TB_INPUT_CHECKBOX Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable to display as check box input tags.
string	*prompt*	IN	The column number in *table* or a string containing the text to display next to the check box. This parameter is required but can have a null ("") value. When *prompt* is null, the value used is the value defined for *namecol*.
string	*namecol*	IN	The column number in *table* or a string containing the input field names.
string	*valuecol*	IN	The column number in *table* or a string containing the input field values. The default is 1.
integer	*rows*	IN	The list of rows in *table* from which to generate the input fields. The default is to use all rows.
integer	*checkedrows*	IN	The list of rows specifying which *rows* of *table* to check. The default is not to check fields.

Examples

Example 1: Generates HTML for three check box input tags

```
@DTW_TB_INPUT_CHECKBOX(Mytable,"3","4","","2 3 4","1 3 4")
```

Results:

```
<INPUT TYPE="CHECKBOX" NAME="link2text" VALUE="1">image2text<BR>
<INPUT TYPE="CHECKBOX" NAME="link3text" VALUE="1" CHECKED>image3text<BR>
<INPUT TYPE="CHECKBOX" NAME="link4text" VALUE="1" CHECKED>image4text<BR>
```

DTW_TB_INPUT_RADIO

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns one or more HTML radio button input tags from a macro table variable.

Format

@DTW_TB_INPUT_RADIO(table, prompt, namecol, valuecol, rows, checkedrows)

Values

Table 76. DTW_TB_INPUT_RADIO Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable to display as radio button input tags.
string	*prompt*	IN	The column number in *table* or a string containing the text to display next to the radio button. Required parameter, but can contain a null ("") value. When *prompt* is null, uses the value of *valuecol*.
string	*namecol*	IN	The column number in *table* or a string containing the input field names.
string	*valuecol*	IN	The column number in *table* or a string containing the input field values.
string	*rows*	IN	The list of rows in *table* from which to generate the input fields. The default is to use all rows.
integer	*checkedrows*	IN	A row number in *table* to display the corresponding radio button as checked. Only one value is allowed.

Examples

Example 1: Generates HTML for three radio button input tags

```
@DTW_TB_INPUT_RADIO(Mytable,"3","Radio4","4","2 3 4","4")
```

Results:

```
<INPUT TYPE="RADIO" NAME="Radio4" VALUE="link2text">image2text<BR>
<INPUT TYPE="RADIO" NAME="Radio4" VALUE="link3text">image3text<BR>
<INPUT TYPE="RADIO" NAME="Radio4" VALUE="link4text" CHECKED>image4text<BR>
```

DTW_TB_INPUT_TEXT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns HTML <INPUT> tags for specified rows in a Net.Data table. For example, Net.Data can generate HTML <INPUT> tags with the VALUE, SIZE, and MAXLENGTH attributes:

```
<INPUT TYPE="TEXT" NAME="someName" VALUE="someValue" SIZE="20" MAXLENGTH="40">
```

Format

@DTW_TB_INPUT_TEXT(table, prompt, namecol, valuecol, size, maxlen, rows)

Values

Table 77. DTW_TB_INPUT_TEXT Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable to display as text input tags.
string	*prompt*	IN	The column number in *table* or a string containing the text to display next to the input field. If *prompt* is null, no text is displayed.
string	*namecol*	IN	The column number in *table* or a string containing the input field names.
string	*valuecol*	IN	The column number in *table* or a string containing the default input field values, which is specified for the VALUE attribute on the INPUT tag. The default is to not generate the VALUE attribute value.
integer	*size*	IN	The number of characters of the input field, which is specified for the SIZE attribute on the INPUT tag. The default is the length of the longest default input value, or 10 if no default input exists.
integer	*maxlen*	IN	The maximum length of an input string, which is specified for the MAXLENTH attribute of the INPUT tag. The default is not to generate the MAXLENGTH attribute value.
integer	*rows*	IN	The list of rows in *table* from which to generate the input fields. The default is to use all rows.

Examples

Example 1: Returns three HTML <INPUT> tags

```
@DTW_TB_INPUT_TEXT(Mytable,"3","3","4","35","40","1 2 3")
```

Results:

```
<P>image1text
<INPUT TYPE="TEXT" NAME="image1text" VALUE="link1text" SIZE="35" MAXLENGTH="40">
<P>image2text
```

```
<INPUT TYPE="TEXT" NAME="image2text" VALUE="link2text" SIZE="35" MAXLENGTH="40">
<P>image3text
<INPUT TYPE="TEXT" NAME="image3text" VALUE="link3text" SIZE="35" MAXLENGTH="40">
```

DTW_TB_LIST

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns an HTML list.

Format

@DTW_TB_LIST(table, listtype, listitem, itemstyle, link_u, image_u)

Values

Table 78. DTW_TB_LIST Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A symbol specifying the macro table variable to display as an HTML list.
string	*listtype*	IN	The type of list to generate. Acceptable values include: DIR MENU OL UL
integer	*listitem*	IN	The column number in *table* containing the list values (the text to go after the <LI> tag). The default is to use the first column.
string	*itemstyle*	IN	A variable or literal string containing a list of HTML elements for the term name values. The default is to use no style tags.
integer	*link_u*	IN	The column number in *table* that contains the URLs for the HTML links. If this value is not specified, no HTML links are generated.
integer	*image_u*	IN	The column number in *table* that contains the URLs for the inline images. If this value is not specified, no inline images are generated.

Examples

Example 1: Generates HTML tags for an ordered list

```
@DTW_TB_LIST(Mytable,"OL","4","TT U","2","1")
```

Results:

```
<TT><U>
<OL>
<LI><A HREF="http://www.mycompany.com/link1.html">
<IMG SRC="http://www.mycompany.com/images/image1.gif" ALT="">link1text</A>
<LI><A HREF="http://www.mycompany.com/link2.html">
<IMG SRC="http://www.mycompany.com/images/image2.gif" ALT="">link2text</A>
<LI><A HREF="http://www.mycompany.com/link3.html">
<IMG SRC="http://www.mycompany.com/images/image3.gif" ALT="">link3text</A>
<LI><A HREF="http://www.mycompany.com/link4.html">
<IMG SRC="http://www.mycompany.com/images/image4.gif" ALT="">link4txt</A>
</OL>
</U></TT>
```

DTW_TB_ROWS

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X			X	X	X

Purpose

Returns the current number of rows in a table.

Format

@DTW_TB_ROWS(table, rows)

@DTW_TB_rROWS(table)

Values

Table 79. DTW_TB_ROWS Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable for which the current number of rows is returned.
integer	*rows*	OUT	A variable that contains the current number of rows in *table*.

Examples

Example 1: Retrieves the current number of rows in the table and assigns the value to *rows*

```
%DEFINE myTable = %TABLE
%DEFINE rows = ""
...
@FillTable()
...
@DTW_TB_ROWS(myTable, rows)
```

Example 2: Retrieves and displays the value for the current number of rows

```
%DEFINE myTable = %TABLE
...
@FillTable()
...
<P>The table value of row 1, column 1 is @DTW_TB_rROWS(myTable, "1", "1").
```

DTW_TB_SELECT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns an HTML SELECT menu.

Format

@DTW_TB_SELECT(table, name, optioncol, size, multiple, rows, selectrows)

Values

Table 80. DTW_TB_SELECT Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable to display as a SELECT field.
string	*name*	IN	The value of the NAME attribute of the SELECT field.
integer	*optioncol*	IN	The column number in *table* with values to use in the OPTION tags of the SELECT field. The default is to use the first column.
integer	*size*	IN	The number of rows in *table* to use for OPTION tags in the SELECT field. The default is to use all the rows.
string	*multiple*	IN	Specifies whether multiple selections are allowed. The default is N, which does not allow multiple selections.
string	*selectedrows*	IN	The row numbers from *table* to use in the SELECT field. The default is to use all the rows.
string	*rows*	IN	The list of rows from table whose OPTION tags are checked. To specify more than one row, you must have the multiple parameter set to Y. The default is to select the first item.

Examples

Generates an HTML SELECT menu with multiple selections

```
@DTW_TB_SELECT(Mytable,"URL6","3","","y","1 2 4","1 4")
```

Results:

```
<SELECT NAME="URL6" SIZE="3" MULTIPLE>
<OPTION SELECTED>image1text
<OPTION>image2text
<OPTION SELECTED>image4text
</SELECT>
```

DTW_TB_TABLE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns an HTML table from a macro table variable.

Format

@DTW_TB_TABLE(table, options, collist, cellstyle, link_u, image_u, url_text, url_style)

Values

Table 81. DTW_TB_TABLE Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A macro table variable to display as an HTML table.
string	*options*	IN	The table attributes inside the TABLE tag. The default is to use no attributes. Valid values include: • BORDER • CELLSPACING • WIDTH
string	*collist*	IN	The column numbers in *table* to use in the HTML table. The default is to use all the columns.
string	*cellstyle*	IN	A list of HTML style elements, such as B and I, to go around text in each TD tag. The default is not to use style tags.
integer	*link_u*	IN	The column number in *table* containing URLs used to create HTML links. You must specify the column in *collist* also. The default is not to generate HTML links.
integer	*image_u*	IN	The column number in *table* containing URLs used to create inline images. You must specify the column in *collist* also. The default is not to generate image tags.
integer	*url_text*	IN	The column number in *table* containing text to display for HTML links or inline images. The default is to use the URL itself.
string	*url_style*	IN	A list of HTML style elements for the text specified in *url_text*. The default is not to generate style tags.

Examples

Example 1: Generates HTML tags for a table with a border and using B (bold) and I (italics) tags

```
@DTW_TB_TABLE(Mytable,"BORDER","4 2 1","i","2","1","4","b")
```

Results:

```
<TABLE BORDER>
<TR>
<TH>TITLE
<TH>LINKURL
<TH>IMAGEURL
<TR>
<TD><i>link1text</i>
<TD><A HREF="http://www.mycompany.com/link1.html"><b>link1text</b></A>
<TD><IMG SRC="http://www.mycompany.com/images/image1.gif" ALT=""><b>link1text</b>
<TR>
<TD><i>link2text</i>
<TD><A HREF="http://www.mycompany.com/link2.html"><b>link2text</b></A>
<TD><IMG SRC="http://www.mycompany.com/images/image2.gif" ALT=""><b>link2text</b>
<TR>
<TD><i>link3text</i>
<TD><A HREF="http://www.mycompany.com/link3.html"><b>link3text</b></A>
<TD><IMG SRC="http://www.mycompany.com/images/image3.gif" ALT=""><b>link3text</b>
</TABLE>
```

DTW_TB_TEXTAREA

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns HTML TEXTAREA tags from a macro table variable.

Format

@DTW_TB_TEXTAREA(table, name, numrows, numcols, valuecol, rows)

Values

Table 82. DTW_TB_TEXTAREA Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A macro table variable to show as a TEXTAREA tag.
string	*name*	IN	The name of the text area.
integer	*numrows*	IN	The height of the text area, specified in rows. The default is the number of rows in *table*.
integer	*numcols*	IN	The width of the text area, specified in columns. The default is the length of the longest row in *table*.
integer	*valuecol*	IN	The column number in *table* whose values are shown in the text area. The default is the first column.
string	*rows*	IN	A list of rows in *table* used to generate the TEXTAREA tag. The default is to use all rows.

Examples

Example 1: Generates HTML TEXTAREA tags and specifies which rows to include

```
@DTW_TB_TEXTAREA(Mytable,"textarea5","3","70","4","1 3 4")
```

Results:

```
<TEXTAREA NAME="textarea5" ROWS="3" COLS="70">
link1text
link3text
link4text
<TEXTAREA>
```

Flat File Interface Functions

The flat file interface (FFI) enables you to open, read, and manipulate data from flat file sources (text files), as well as store data in flat files.

Flat File Interface Delimiters

In order to improve performance, you can keep the Net.Data tabular output from a series of SQL requests in a flat file. You can retrieve the flat file in subsequent requests, instead of re-issuing the SQL requests.

Net.Data flat files can be created from Net.Data tables and Net.Data tables can be built from flat files. In order to make the transformations between the tables and flat files, you must define the mapping between columns in a table and records in a flat file. Delimiters provide a method for defining how portions of records in a flat file can be separated and mapped to columns in a table, and how columns in a table can be mapped to records in a flat file.

There are two types of delimiters:

New-line character (ASCIITEXT)
Use this transformation when your table is made up of one column. Net.Data maps each record in the corresponding flat file onto a single row in the table. In this case, the regular new-line character which separates records in the flat file is the only delimiter used.

New-line character and delimiter string (DELIMITED)
Use this transformation when your table is made up of multiple columns. When Net.Data creates a flat file record from a row in a table, it places the delimiter string as a separator between the items. When Net.Data rebuilds a table from a flat file, it uses the delimiter string to determine how much of each row to place in a column of the table. In this case, the regular new-line character separates the records in the flat file that correspond to rows in the table, and the delimiter string separates the items within a single record.

You can use the DTWF_SEARCH function to retrieve certain records held in a flat file that was built from a Net.Data table. Specify a string in DTWF_SEARCH to return all the records that contain the string in the flat file as rows in a Net.Data table.

Flat File Interface Functions

- “DTWF_WRITE” on page 210

DTWF_APPEND

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Writes the contents of a table variable to the end of a file.

Format

@DTWF_APPEND(filename, transform, delimiter, table, retry, rows)

Values

Table 83. DTWF_APPEND Parameters

Data Type	**Parameter**	**Use**	**Description**
string	*filename*	INOUT	The name of the file to which the variable's contents are being added. On successful completion of the call, this parameter returns the fully qualified file name.
string	*transform*	IN	The format of the file: • ASCIITEXT - writes the table to the file with a new-line character between column values and ignores the *delimiter* parameter. • DELIMITED - writes the table to the file with the delimiter specified in the *delimiter* parameter.
string	*delimiter*	IN	A character string to indicate the ends of values. This parameter is case sensitive. Ignored if *transform* is ASCIITEXT.
table	*table*	IN	The table variable from which the records are read.
integer	*retry*	IN	The number of times to retry if the file cannot be appended to immediately. The default is not to retry.
integer	*rows*	IN	The maximum number of rows from *table* to append. The default is to append all the rows. Specifying 0 appends all rows.

Examples

Example 1:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
%}
@DTWF_APPEND(myFile, "DELIMITED", " ;", myTable)
```

Example 2:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
%}
@DTWF_APPEND(myFile, "ASCIITEXT", " ;", myTable)
```

Example 3:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
%}
@DTWF_APPEND(myFile, "ASCIITEXT", " ;", myTable, "0", "10")
```

DTWF_CLOSE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Closes a file opened by DTWF_OPEN.

Format

@DTWF_CLOSE(filename, retry)

Values

Table 84. DTWF_CLOSE Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file to close. On successful completion of the call, this parameter returns the fully qualified file name.
integer	*retry*	IN	The number of times to retry if the file cannot be closed immediately. The default is not to retry.

Examples

Example 1:

```
@DTWF_CLOSE(myFile, "5")
```

DTWF_DELETE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Deletes records from a file. (Does not delete empty files.)

Format

@DTWF_DELETE(filename, transform, delimiter, retry, rows, startrow)

Values

Table 85. DTW_DELETE Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file whose records are to be deleted. On successful completion of the call, this parameter returns the fully qualified file name.
string	*transform*	IN	The format of the file: • ASCIITEXT - writes the table to the file with a new-line character between column values and ignores the *delimiter* parameter. • DELIMITED - writes the table to the file with the delimiter specified in the *delimiter* parameter.
string	*delimiter*	IN	A character string to indicate the ends of values. This parameter is case sensitive. Ignored if *transform* is ASCIITEXT.
integer	*retry*	IN	The number of times to retry if the records cannot be deleted immediately. The default is not to retry.
integer	*rows*	IN	The maximum number of rows to delete. The default is to delete all the rows. Specifying 0 deletes all rows.
integer	*startrow*	INOUT	The row number from which to begin deleting. A value of 1 means to begin deleting at the first row. If this value is greater than the number of rows in the file, the value is changed to the last record and returned as an error. The default is to start at 1.

Examples

Example 1:

```
%DEFINE {
   myFile = "c:/private/myfile"
   myTable = %TABLE
   myWait = "5000"
   myRows = "2"
%}
@DTWF_DELETE(myFile, "Delimited", "|", myWait, myRows)
```

Example 2:

```
%DEFINE {
   myFile = "c:/private/myfile"
   myTable = %TABLE
   myStart = "1"
   myRows = "2"
%}
@DTWF_DELETE(myFile, "Asciitext", "|", "0", myRows, myStart)
```

DTWF_INSERT

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Inserts records into a file.

Format

@DTWF_INSERT(filename, transform, delimiter, table, retry, rows, startrow)

Values

Table 86. DTWF_INSERT Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file to which records are inserted. On successful completion of the call, this parameter returns the fully qualified file name.
string	*transform*	IN	The format of the file: • ASCIITEXT - writes the table to the file with a new-line character between column values and ignores the *delimiter* parameter. • DELIMITED - writes the table to the file with the delimiter specified in the *delimiter* parameter.
string	*delimiter*	IN	A character string to indicate the ends of values. This parameter is case sensitive. Ignored if *transform* is ASCIITEXT.
table	*table*	IN	The table variable from which records are inserted into the file.
integer	*retry*	IN	The number of times to retry if the file cannot be written to immediately. The default is not to retry.
integer	*rows*	IN	The maximum number of rows to insert from *table*. The default is to insert all the rows. A value of 0 inserts all the rows.
integer	*startrow*	INOUT	The row number from which to begin inserting. Specifying 1 means to begin inserting at the first row. If this value is greater than the number of rows in the file, the value is changed to the last record and returned as an error. The default is to start at 1.

Examples

Example 1:

```
%DEFINE {
   myFile = "c:/private/myfile"
   myTable = %TABLE
   myWait = "3000"
%}
@DTWF_INSERT(myFile, "Delimited", "|", myTable, myWait)
```

Example 2:

```
%DEFINE {
   myFile = "c:/private/myfile"
   myTable = %TABLE
   myStart = "1"
   myRows = "2"
%}
@DTWF_INSERT(myFile, "Asciitext", "|", myTable, "0", myRows, myStart)
```

DTWF_OPEN

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Explicitly opens a file. DTWF_OPEN keeps the file open, otherwise, the file is closed after each flat file operation.

Performance tip: Use DTWF_OPEN to reduce the number of times a file is open.

The file is left open until it is closed using DTWF_CLOSE or macro processing ends.

Format

@DTWF_OPEN(filename, mode, retry)

Values

Table 87. DTWF_OPEN Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file to open. On successful completion of the call, this parameter returns the fully qualified file name.
string	*mode*	IN	The type of access requested: • r - opens an existing file for reading. • w - creates a file for writing. (Destroys existing file of same name, if it exists.) • a - opens a file for appending. Net.Data creates the file if it is not found. • r+ - opens an existing file for reading and writing. • w+ - creates a file for reading and writing. (Destroys existing file of same name, if it exists.) • a+ - opens a file in append mode for reading or appending. Net.Data creates the file if it is not found.
integer	*retry*	IN	The number of times to retry if the file cannot be opened immediately. The default is not to retry.

Examples

Example 1:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myMode = "r+"
%}
@DTWF_OPEN(myFile, myMode, "1000")
```

DTWF_READ

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Reads records from a file into a table variable.

Format

@DTWF_READ(filename, transform, delimiter, table, retry, rows, startrow, columns)

Values

Table 88. DTWF_READ Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file whose records are read into a table variable. On successful completion of the call, this parameter returns the fully qualified file name.
string	*transform*	IN	The format of the file: • ASCIITEXT - writes the table to the file with a new-line character between column values and ignores the *delimiter* parameter. • DELIMITED - writes the table to the file with the delimiter specified in the *delimiter* parameter.
string	*delimiter*	IN	A character string to indicate the ends of values. This parameter is case sensitive. Ignored if *transform* is ASCIITEXT.
table	*table*	OUT	The table variable into which the file records are read.
integer	*retry*	IN	The number of times to retry if the file cannot be read immediately. The default is not to retry.
integer	*rows*	INOUT	The maximum number of file records to read into table. The default is to read all the records, or until the table is full. A value of 0 means to read until the end of the file. The number of rows in the resulting table is returned.
integer	*startrow*	IN	The record in the file from which to start reading. The default is to start reading at the first record.
integer	*columns*	OUT	Returns the number of columns in the table.

Examples

Example 1:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
  myWait = "1000"
%}
@DTWF_READ(myFile, "DELIMITED", ";", myTable, myWait)
```

Example 2:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
  myWait = "0"
  myRows = "0"
  myStartrow = "1"
  myColumns = ""
%}
@DTWF_READ(myFile, "DELIMITED", ";", myTable, myWait, myRows,
             myStartrow, myColumns)
```

Example 3:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
%}
@DTWF_READ(myFile, "ASCIITEXT", ";", myTable, myColumns)
@DTW_TB_TABLE(myTable,"BORDER","")
```

DTWF_REMOVE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Deletes an entire file.

Format

@DTWF_REMOVE(filename, retry)

Values

Table 89. DTW_REMOVE Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file to delete. On successful completion of the call, this parameter returns the fully qualified file name.
integer	*retry*	IN	The number of times to retry if the file cannot be deleted immediately. The default is not to retry.

Examples

Example 1:

```
%DEFINE myFile = "c:/private/myfile"
@DTWF_REMOVE(myFile)
```

Example 2:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myWait = "2000"
%}
@DTWF_REMOVE(myFile, myWait)
```

DTWF_SEARCH

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Returns result of a string search to a table variable.

Format

@DTWF_SEARCH(filename, transform, delimiter, table, searchFor, retry, rows, startrow)

Values

Table 90. DTWF_SEARCH Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file to search. On successful completion of the call, this parameter returns the fully qualified file name.
string	*transform*	IN	The format of the file: • ASCIITEXT - writes the table to the file with a new-line character between column values and ignores the *delimiter* parameter. • DELIMITED - writes the table to the file with the delimiter specified in the *delimiter* parameter.
string	*delimiter*	IN	A character string to indicate the ends of values. This parameter is case sensitive. Ignored if *transform* is ASCIITEXT.
table	*table*	OUT	The table variable into which the search results are placed. Three columns are returned if *transform* is DELIMITED: • The row in which the match was found. • The column in which the match was found. • The matching column from the file.
string	*searchFor*	IN	The string of characters to search for.
integer	*retry*	IN	The number of times to retry if the file cannot be searched immediately. The default is not to retry.
integer	*rows*	INOUT	The maximum number of rows to read into *table*. The default is to read all the rows or until *table* is full. Specifying 0 reads to the end of the file. The number of rows in the resulting table is returned by this parameter.
integer	*startrow*	IN	The record in the file to start searching from. The default is 1, which begins the search at the first record.

Examples

Example 1:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
  myWait = "1000"
  mySearch = "0123456789abcdef"
@DTWF_SEARCH(myFile, "DELIMITED", ";",
               myTable, mySearch, myWait)
```

Example 2:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
  mySearch = "answer:"
  myWait = "0"
  myRows = "0"
  myStartrow = "1"
%}
@DTWF_SEARCH(myFile, "DELIMITED", ";", myTable,
               mySearch, myWait, myRows, myStartrow)
```

DTWF_UPDATE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Updates records of a file from a table variable.

Format

@DTWF_UPDATE(filename, transform, delimiter, table, retry, rows, startrow)

Values

Table 91. DTWF_UPDATE Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file whose records are updated from a table variable. On successful completion of the call, this parameter returns the fully qualified file name.
string	*transform*	IN	The format of the file: • ASCIITEXT - writes the table to the file with a new-line character between column values and ignores the *delimiter* parameter. • DELIMITED - writes the table to the file with the delimiter specified in the *delimiter* parameter.
string	*delimiter*	IN	A character string to indicate the ends of values. This parameter is case sensitive. Ignored if *transform* is ASCIITEXT.
table	*table*	IN	The table variable from which the file records are updated.
integer	*retry*	IN	The number of times to retry if the file cannot be written to immediately. The default is not to retry.
integer	*rows*	IN	The maximum number of records to be updated from *table*. The default is to update all the records. A value of 0 means to update all rows in the file.
integer	*startrow*	INOUT	The first file record to update. The default is 1, which means to start updating at the beginning of the file. If the value is greater than the number of records in a file, the value is changed to indicate the number of the last record in the file and an error is returned.

Examples

Example 1:

```
%DEFINE {
   myFile = "c:/private/myfile"
   myTable = %TABLE
   myWait = "1500"
   myRows = "2"
%}
@DTWF_UPDATE(myFile, "Delimited", "|", myTable, myWait, myRows)
```

Example 2:

```
%DEFINE {
   myFile = "c:/private/myfile"
   myTable = %TABLE
   myStart = "1"
   myRows = "2"
%}
@DTWF_UPDATE(myFile, "Asciitext", "|", myTable, "0", myRows, myStart)
```

DTWF_WRITE

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X	X	X	X	X	X

Purpose

Writes the contents of a table variable to a file.

Format

@DTWF_WRITE(filename, transform, delimiter, table, retry, rows, startrow)

Values

Table 92. DTWF_WRITE Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file the records of the table variable are written to. On successful completion of the call, this parameter returns the fully qualified file name.
string	*transform*	IN	The format of the file: • ASCIITEXT - writes the table to the file with a new-line character between column values and ignores the *delimiter* parameter. • DELIMITED - writes the table to the file with the delimiter specified in the *delimiter* parameter.
string	*delimiter*	IN	A character string to indicate the ends of values. This parameter is case sensitive. Ignored if *transform* is ASCIITEXT.
table	*table*	IN	The table variable used to export rows to the file.
integer	*retry*	IN	The number of times to retry if the file cannot be written to immediately. The default is to not retry.
integer	*rows*	IN	The maximum number of file records to write. The default is to write the entire table. A value of 0 means to write all records to the end of the file.
integer	*startrow*	INOUT	The record number to start writing to in the file. The default is 1, which means to start at the first record. If a value beyond the end of the file is specified, the last row of the file is returned with an error.

Examples

Example 1:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
%}
@DTWF_WRITE(myFile, "DELIMITED", ";", myTable)
```

Example 2:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
%}
@DTWF_WRITE(myFile, "ASCIITEXT", ";", myTable, "5000")
```

Example 3:

```
%DEFINE {
  myFile = "c:/private/myfile"
  myTable = %TABLE
%}
@DTWF_WRITE(myFile, "ASCIITEXT", ";", myTable, "5000", "10", "50")
```

Web Registry Functions

A Web registry is a file with a key maintained by Net.Data to allow you to add, retrieve, and delete entries easily. You can create multiple Net.Data Web registries on a single system. Each registry has a name and can contain multiple entries. Net.Data provides functions to maintain registries and the entries they contain.

- “DTWR_ADDENTRY” on page 213
- “DTWR_CLEARREG” on page 214
- “DTWR_CREATEREG” on page 215
- “DTWR_DELENTRY” on page 216
- “DTWR_DELREG” on page 217
- “DTWR_LISTREG” on page 218
- “DTWR_LISTSUB” on page 219
- “DTWR_RTVENTRY” on page 220
- “DTWR_UPDATEENTRY” on page 221

Restriction: Do not use asterisks (*) for the *registry*, *registryVariable*, and *registryData* parameters when using OS/2.

DTWR_ADDENTRY

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Adds an entry to a Web registry.

Format

@DTWR_ADDENTRY(registry, registryVariable, registryData, index)

Values

Table 93. DTWR_ADDENTRY Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to which the entry is added.
string	*registryVariable*	IN	The value of the *registryVariable* string portion of the registry entry to add.
string	*registryData*	IN	The value of the *registryData* string portion of the registry entry to add.
string	*index*	IN	The value of the index portion of the *registryVariable* string in an indexed entry to add. This parameter is optional. If specified, an indexed entry is added to the specified registry.

Examples

Example 1:

```
@DTWR_ADDENTRY("Myregistry", "Jones", "http://Advantis.com/˜Jones/webproj")
```

Example 2:

```
@DTWR_ADDENTRY("URLLIST", "SMITH", "http://www.software.ibm.com/",
     "WORK_URL,")
```

DTWR_CLEARREG

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Clears entries from a Web registry.

Format

@DTWR_CLEARREG(registry)

Values

Table 94. DTWR_CLEARREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to clear.

Examples

Example 1:

```
@DTWR_CLEARREG("Myregistry")
```

DTWR_CREATEREG

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Creates a new Web registry.

Re

Format

@DTWR_CREATEREG(registry, security)

Values

Table 95. DTWR_CREATEREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to create. **Restriction:** Do not use special characters such as the asterisk (*) and the backslash (\) in Web registry names.
string	*security*	IN	The type of security with which to create *registry*. On UNIX operating systems, the default security is the same as the directory where the registry is created. Specify security for the three security groups: user, group, and public. R gives read permission, W gives write permission, and X give execute permission. For example, to give all three groups full authority, specify `*RWX, *RWX, *RWX` for this parameter. .

Examples

Example 1:

```
@DTWR_CREATEREG("myRegistry")
```

Example 2:

```
@DTWR_CREATEREG("URLLIST", "*RWX, *RWX, *R")
```

DTWR_DELENTRY

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Deletes an entry from a Web registry.

Format

@DTWR_DELENTRY(registry, registryVariable, index)

Values

Table 96. DTWR_DELENTRY Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry from which the entry is removed.
string	*registryVariable*	IN	The value of the *registryVariable* string portion of the entry to remove.
string	*index*	IN	The value of the index portion of the *registryVariable* string in an indexed entry. This is an optional parameter. If specified, the indexed entry is removed from the registry.

Examples

Example 1:

```
@DTWR_DELENTRY("Myregistry", "Jones")
```

Example 2:

```
@DTWR_DELENTRY("URLLIST", "SMITH", "WORK_URL")
```

DTWR_DELREG

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Deletes a Web registry

Format

@DTWR_DELREG(registry)

Values

Table 97. DTWR_DELREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to delete.

Examples

Example 1:

```
@DTWR_DELREG("Myregistry")
```

DTWR_LISTREG

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Lists an entire Web registry. DTWR_LISTREG returns information about the registry entries in an OUT table variable passed by the user. The table variable is defined in the user macro before being passed as a parameter to the FUNCTION block for the LISTREG registry operation.

If the user defined the table variable using the ALL option for the maximum number of rows for the table, this operation lists all available registry entries in the table, one for each table row. On the other hand if the user specified a value X for the maximum number of table rows, then if there are more then X entries in the specified registry only the first X entries are listed and an error code is sent back to indicate that only a partial listing could be done because not enough table rows were available to list additional entries. All registry entries are listed if the value X exceeds the number of available entries in the specified registry.

There are always 2 columns in the table. The Column headers for the table are set to ″REGISTRY_VARIABLE″ and ″REGISTRY_DATA″ by the Web Registry language environment.

Format

@DTWR_LISTREG(registry, registryTable)

Values

Table 98. DTWR_LISTREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to list.
string	*registryTable*	OUT	The name of the table variable in which the registry entries are placed.

Examples

Example 1:

```
%DEFINE RegistryTable = %TABLE(ALL)

@DTWR_LISTREG("URLLIST", RegistryTable)
```

DTWR_LISTSUB

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
							X

Purpose

Lists immediate subkey entries in a Web registry. DTWR_LISTSUB returns information about the registry entries in an OUT table parameter passed by the user. The table variable is defined in the macro before being passed as a parameter to the LISTSUB registry operation.

If the user has defined the table variable using the ALL option for the maximum number of rows for the table, this operation lists all available registry entries in the table, one for each table row. On the other hand, if the user specified a value X for the maximum number of table rows then if there are more then X entries in the specified registry only the first X entries are listed and an error code is sent back to indicate that only a partial listing could be done because not enough table rows are available to list additional entries. All registry entries are listed if the value X exceeds the number of available entries in the specified registry. The number of columns in the table is always one.

The column header for the table is set to ″REGISTRY_SUBKEY″.

This function is only valid on operating system that are compatible Windows 95 System Registries.

Format

@DTWR_LISTSUB(registry, registryTable)

Values

Table 99. DTWR_LISTSUB Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to list.
string	*registryTable*	OUT	The name of the table variable in which the registry entries are placed.

Examples

Example 1:

```
%DEFINE RegistryTable = %TABLE(ALL)

@DTWR_LISTSUB("URLLIST", RegistryTable)
```

DTWR_RTVENTRY

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Retrieves the registryData string from a Web registry entry.

Format

@DTWR_RTVENTRY(registry, registryVariable, registryData, index)

@DTWR_rRTVENTRY(registry, registryVariable, index)

Values

Table 100. DTWR_RTVENTRY Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry with entries to retrieve.
string	*registryVariable*	IN	The value of the *registryVariable* string portion of the registry entry whose registryData string is retrieved.
string	*registryData*	OUT	Returns the value of the *registryData* string portion of the registry entry that matches the *registryVariable*.
string	*index*	IN	The value of the index portion of the *registryVariable* string in an indexed entry whose *registryData* string is returned. This is an optional parameter. If specified, the *registryData* string of the indexed entry is returned.

Examples

Example 1:

```
%DEFINE RegistryData = ""
@DTWR_RTVENTRY("Myregistry", "Jones", RegistryData)
```

Example 2:

```
@DTWR_RTVENTRY("URLLIST", "SMITH", RegistryData, "WORK_URL")
```

Example 3:

```
@DTWR_rRTVENTRY("Myregistry", "Jones")
```

Example 4:

```
@DTWR_rRTVENTRY("URLLIST", "SMITH", "WORK_URL")
```

DTWR_UPDATEENTRY

AIX	HP-UX	OS/2	OS/390	OS/400	SCO	SUN	Win NT
X	X	X		X	X	X	X

Purpose

Replaces the existing *registryData* string value for the specified registry entry with the new value specified by the caller. The *registerVariable* string cannot be changed.

Format

@DTWR_UPDATEENTRY(registry, registryVariable, newData, index)

Values

Table 101. DTWR_UPDATEENTRY Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry with the entry to update.
string	*registryVariable*	IN	The value of the *registryVariable* string portion of the registry entry to update.
string	*newData*	IN	The new value for the *registryData* string portion of the registry entry to update.
string	*index*	IN	The value of the index portion of the *registryVariable* string in an indexed entry to update. This is an optional parameter. If specified, the indexed entry is updated.

Examples

Example 1:

```
@DTWR_UPDATEENTRY("Myregistry", "Jones", "http://advantis.com/~Jones/personal")
```

Example 2:

```
@DTWR_UPDATEENTRY("URLLIST", "SMITH", "http://www.software.ibm.com/personal", "WORK_URL")
```

Appendix A. DB2 WWW Connection

If you have DB2 WWW Connection, you can run your existing applications with Net.Data. We recommend updating your applications to take advantage of Net.Data Version 2 features.

The DB2 WWW language constructs are:

- "EXEC_SQL"
- "HTML_INPUT"
- "HTML_REPORT"
- "SQL"
- "SQL_MESSAGE" on page 224
- "SQL_REPORT" on page 224
- "SQL_CODE" on page 224

EXEC_SQL

This language construct calls an SQL block. We recommend calling SQL statements as functions instead. See "FUNCTION Block" on page 16 for more information.

HTML_INPUT

This language construct is the same as an HTML block named INPUT. See "HTML Block" on page 24 for more information.

HTML_REPORT

This language construct is the same as an HTML block named REPORT. See "HTML Block" on page 24 for more information.

SQL

This language construct is equivalent to a function called with FUNCTION(DTW_SQL) in Net.Data.

It can contain SQL_REPORT and SQL_MESSAGE statements, which are also from DB2 WWW Connection. DB2 WWW Connection does not support named %SQL blocks.

Examples:

Example 1: A DB2 WWW Connection macro

```
%SQL{
UPDATE $(dbtbl) SET URL='$(URL)' WHERE ID=$(ID)
%SQL_MESSAGE{
100: "<B>The selected URL no longer exists in the table</B>." : continue
%}
%}

%HTML_INPUT{
<HTML>
...
%EXEC_SQL
</HTML>
%}

%HTML_REPORT{
<HTML>
...
</HTML>
%}
```

Example 1: An equivalent Net.Data macro

```
%FUNCTION(DTW_SQL) URLquery(){
UPDATE $(dbtbl) SET URL='$(URL)' WHERE ID=$(ID)
%MESSAGE{
100: "<B>The selected URL no longer exists in the table</B>." : continue
%}
%}

%HTML(INPUT){
<HTML>
...
@URLquery
</HTML>
%}

%HTML(REPORT){
<HTML>
...
</HTML>
%}
```

SQL_MESSAGE

This language construct is equivalent to the Net.Data MESSAGE statement. See “MESSAGE Block” on page 42 for an example.

SQL_REPORT

This language construct is equivalent to the Net.Data REPORT statement. See “REPORT Block” on page 47for an example.

SQL_CODE

This language construct is from DB2 WWW connection and supported by Net.Data for compatibility. It is equivalent to “RETURN_CODE” on page 110.

Appendix B. Net.Data Operating System Reference

Not all Net.Data features are supported on each operating system. This section shows which features are supported for your operating system. An **X** indicates the feature is supported.

Some of the features listed here were not yet available at general availability.

Table 102. Net.Data Language Environments

Language Environment	**AIX**	**HP**	**OS/2**	**OS/390**	**OS/400**	**SCO**	**SUN**	**Win NT**
Default	X	X	X	X	X	X	X	X
Flat File Interface	X	X	X	X	X	X	X	X
IMS Web	X			X				X
Java Applets	X	X	X	X		X	X	X
Java Applications	X		X				X	X
ODBC	X	X	X	X		X	X	X
Oracle	X							X
Perl	X		X	X				X
REXX	X		X	X	X	X	X	X
SQL	X	X	X	X	X	X	X	X
Sybase	X							X
System	X	X	X	X	X	X	X	X
Web Registry	X	X	X		X	X	X	X

Table 103. Net.Data Stored Procedure Data Types

Data Type	**AIX**	**HP**	**OS/2**	**OS/390**	**OS/400**	**SCO**	**SUN**	**Win NT**
BIGINT	X	X	X			X	X	X
BLOB	X	X	X			X	X	X
CHAR	X	X	X	X	X	X	X	X
CLOB	X	X	X			X	X	X
DATE	X	X	X		X	X	X	X
DBCLOB	X	X	X			X	X	X
DECIMAL	X	X	X	X	X	X	X	X
DOUBLE	X	X	X	X	X	X	X	X
DOUBLEPRECISION	X	X	X	X	X	X	X	X
FLOAT	X	X	X	X	X	X	X	X
INTEGER	X	X	X	X	X	X	X	X
GRAPHIC	X	X	X	X	X	X	X	X
LONGVARCHAR	X	X	X		X	X	X	X
LONGVARGRAPHIC	X	X	X		X	X	X	X
SMALLINT	X	X	X	X	X	X	X	X
TIME	X	X	X		X	X	X	X
TIMESTAMP	X	X	X		X	X	X	X
VARCHAR	X	X	X	X	X	X	X	X

Table 103. Net.Data Stored Procedure Data Types (continued)

Data Type	AIX	HP	OS/2	OS/390	OS/400	SCO	SUN	Win NT
VARGRAPHIC	X	X	X	X	X	X	X	X

Table 104. Net.Data Configuration Variables

Configuration Variable	AIX	HP	OS/2	OS/390	OS/400	SCO	SUN	Win NT
CACHE_MACHINE	X							X
CACHE_PORT	X							X
DefaultDBCp				X				
DB2INSTANCE	X	X	X			X	X	X
DB2MSGS				X				
DB2PLAN				X				
DB2SSID				X				
DSNAOINI				X				
DTW_CM_PORT	X	X	X			X	X	X
DTW_INST_DIR	X	X	X			X	X	X
DTW_LOG_DIR	X	X	X			X	X	X
DTW_MBMODE	X	X	X	X		X	X	X
DTW_OPTIMIZE_MATH	X	X	X			X	X	X
DTW_REMOVE_WS				X				
DTW_SMTP_SERVER	X	X	X			X	X	X
DTW_SQL_ISOLATION					X			
DTW_SQL_NAMING_MODE					X			
DTWR_CLOSE_REGISTRIES					X			
LOGIN	X	X	X		X	X	X	X
PASSWORD	X	X	X		X	X	X	X

Table 105. Net.Data Variables

Variable	AIX	HP	OS/2	OS/390	OS/400	SCO	SUN	Win NT
ALIGN	X	X	X	X	X	X	X	X
DATABASE	X	X	X		X	X	X	X
DB_CASE	X	X	X	X	X	X	X	X
DB2PLAN				X				
DB2SSID				X				
DTW_APPLET_ALTTEXT	X	X	X	X		X	X	X
DTW_CURRENT_FILENAME	X	X	X	X	X	X	X	X
DTW_CURRENT_LAST_MODIFIED	X	X	X	X	X	X	X	X
DTW_DEFAULT_MESSAGE					X			
DTW_DEFAULT_REPORT	X	X	X	X	X	X	X	X
DTW_EDIT_CODES					X			
DTW_HTML_TABLE	X	X	X	X	X	X	X	X
DTW_LOG_LEVEL	X	X	X			X	X	X
DTW_MACRO_FILENAME	X	X	X	X	X	X	X	X

Table 105. Net.Data Variables (continued)

Variable	**AIX**	**HP**	**OS/2**	**OS/390**	**OS/400**	**SCO**	**SUN**	**Win NT**
DTW_MACRO_LAST_MODIFIED	X	X	X	X	X	X	X	X
DTW_MBMODE	X	X	X	X		X	X	X
DTW_MP_PATH	X	X	X	X	X	X	X	X
DTW_MP_VERSION	X	X	X	X	X	X	X	X
DTW_PRINT_HEADER	X	X	X	X	X	X	X	X
DTW_REMOVE_WS	X	X	X	X	X	X	X	X
DTW_SAVE_TABLE_IN	X	X	X	X	X	X	X	X
DTW_SET_TOTAL_ROWS	X	X	X		X	X	X	X
LOCATION				X				
LOGIN	X	X	X		X	X	X	X
Nn	X	X	X	X	X	X	X	X
NLIST	X	X	X	X	X	X	X	X
NULL_RPT_FIELD					X			
NUM_COLUMNS	X	X	X	X	X	X	X	X
NUM_ROWS					X			
PASSWORD	X	X	X		X	X	X	X
RETURN_CODE	X	X	X	X	X	X	X	X
ROW_NUM	X	X	X	X	X	X	X	X
RPT_MAX_ROWS	X	X	X	X	X	X	X	X
SHOWSQL	X	X	X	X	X	X	X	X
SQL_CODE	X	X	X	X	X	X	X	X
SQL_STATE	X	X	X	X	X	X	X	X
START_ROW_NUM	X	X	X		X	X	X	X
TOTAL_ROWS	X	X	X		X	X	X	X
TRANSACTION_SCOPE	X	X	X	X	X	X	X	X
V_columnName	X	X	X	X	X	X	X	X
VLIST	X	X	X	X	X	X	X	X
Vn	X	X	X	X	X	X	X	X

Table 106. Net.Data Functions

Function	**AIX**	**HP**	**OS/2**	**OS/390**	**OS/400**	**SCO**	**SUN**	**Win NT**
DTW_ADD	X	X	X	X	X	X	X	X
DTW_ADDQUOTE	X	X	X	X	X	X	X	X
DTW_ASSIGN	X	X	X	X	X	X	X	X
DTW_CACHE_PAGE	X							X
DTW_CONCAT	X	X	X	X	X	X	X	X
DTW_DATE	X	X	X	X	X	X	X	X
DTW_DELSTR	X	X	X	X	X	X	X	X
DTW_DELWORD	X	X	X	X	X	X	X	X
DTW_DIVIDE	X	X	X	X	X	X	X	X

Table 106. Net.Data Functions (continued)

Function	AIX	HP	OS/2	OS/390	OS/400	SCO	SUN	Win NT
DTW_DIVREM	X	X	X	X	X	X	X	X
DTW_EXIT	X	X	X			X	X	X
DTW_FORMAT	X	X	X	X	X	X	X	X
DTW_GETCOOKIE	X	X	X			X	X	X
DTW_GETENV	X	X	X	X	X	X	X	X
DTW_GETINIDATA	X	X	X	X	X	X	X	X
DTW_HTMLENCODE	X	X	X	X	X	X	X	X
DTW_INSERT	X	X	X	X	X	X	X	X
DTW_INTDIV	X	X	X	X	X	X	X	X
DTW_LASTPOS	X	X	X	X	X	X	X	X
DTW_LENGTH	X	X	X	X	X	X	X	X
DTW_LOWERCASE	X	X	X	X	X	X	X	X
DTW_MULTIPLY	X	X	X	X	X	X	X	X
DTW_POS	X	X	X	X	X	X	X	X
DTW_POWER	X	X	X	X	X	X	X	X
DTW_QHTMLENCODE	X	X	X	X	X	X	X	X
DTW_REVERSE	X	X	X	X	X	X	X	X
DTW_SENDMAIL	X	X	X			X	X	X
DTW_SETCOOKIE	X	X	X			X	X	X
DTW_SETENV	X	X	X	X	X	X	X	X
DTW_STRIP	X	X	X	X	X	X	X	X
DTW_SUBSTR	X	X	X	X	X	X	X	X
DTW_SUBTRACT	X	X	X	X	X	X	X	X
DTW_SUBWORD	X	X	X	X	X	X	X	X
DTW_TB_COLS	X	X	X		X	X	X	X
DTW_TB_DLIST	X	X	X	X	X	X	X	X
DTW_TB_DUMPH	X	X	X	X	X	X	X	X
DTW_TB_DUMPV	X	X	X	X	X	X	X	X
DTW_TB_GETN	X	X	X		X	X	X	X
DTW_TB_GETV	X	X	X		X	X	X	X
DTW_TB_HTMLENCODE	X	X	X	X	X	X	X	X
DTW_TB_INPUT_CHECKBOX	X	X	X	X	X	X	X	X
DTW_TB_INPUT_RADIO	X	X	X	X	X	X	X	X
DTW_TB_INPUT_TEXT	X	X	X	X	X	X	X	X
DTW_TB_LIST	X	X	X	X	X	X	X	X
DTW_TB_ROWS	X	X	X		X	X	X	X
DTW_TB_SELECT	X	X	X	X	X	X	X	X
DTW_TB_TABLE	X	X	X	X	X	X	X	X
DTW_TB_TEXTAREA	X	X	X	X	X	X	X	X
DTW_TIME	X	X	X	X	X	X	X	X

Table 106. Net.Data Functions (continued)

Function	**AIX**	**HP**	**OS/2**	**OS/390**	**OS/400**	**SCO**	**SUN**	**Win NT**
DTW_TRANSLATE	X	X	X	X	X	X	X	X
DTW_UPPERCASE	X	X	X	X	X	X	X	X
DTW_URLESCSEQ	X	X	X	X	X	X	X	X
DTW_WORD	X	X	X	X	X	X	X	X
DTW_WORDINDEX	X	X	X	X	X	X	X	X
DTW_WORDLENGTH	X	X	X	X	X	X	X	X
DTW_WORDPOS	X	X	X	X	X	X	X	X
DTW_WORDS	X	X	X	X	X	X	X	X
DTWF_APPEND	X	X	X	X	X	X	X	X
DTWF_CLOSE	X	X	X	X	X	X	X	X
DTWF_DELETE	X	X	X	X	X	X	X	X
DTWF_INSERT	X	X	X	X	X	X	X	X
DTWF_OPEN	X	X	X	X	X	X	X	X
DTWF_READ	X	X	X	X	X	X	X	X
DTWF_REMOVE	X	X	X	X	X	X	X	X
DTWF_SEARCH	X	X	X	X	X	X	X	X
DTWF_UPDATE	X	X	X	X	X	X	X	X
DTWF_WRITE	X	X	X	X	X	X	X	X
DTWR_ADDENTRY	X	X	X		X	X	X	X
DTWR_CLEARREG	X	X	X		X	X	X	X
DTWR_CREATEREG	X	X	X		X	X	X	X
DTWR_DELENTRY	X	X	X		X	X	X	X
DTWR_DELREG	X	X	X		X	X	X	X
DTWR_LISTREG	X	X	X		X	X	X	X
DTWR_LISTSUB	X	X	X			X	X	X
DTWR_RTVENTRY	X	X	X		X	X	X	X
DTWR_UPDATEENTRY	X	X	X		X	X	X	X

Table 107. Net.Data Interfaces

Interface Type	**AIX**	**HP**	**OS/2**	**OS/390**	**OS/400**	**SCO**	**SUN**	**Win NT**
FastCGI	X							
CGI	X	X	X	X	X	X	X	X
Java Beans								X
Internet Connection API (ICAPI)	X		X	X				X
Internet Server API (ISAPI)								X
Live Connection	X	X	X				X	X
Lotus Domino Go Web Server (GWAPI)	X		X	X				X
Netscape API (NSAPI)	X						X	X
Servlets	X							X

Table 108. Net.Data Tools

Tool	**AIX**	**HP**	**OS/2**	**OS/390**	**OS/400**	**SCO**	**SUN**	**Win NT**
Administration Tool	X		X					X
NetObjects Fusion Plug-ins								X
Wizards	X	X	X			X	X	X

Appendix C. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
DataJoiner
DB2
Domino
IBM
IMS
Lotus
MVS
Net.Data
OS/2
OS/390
OS/400

The following terms are trademarks of other companies as follows:

Java and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT®, and the Windows 95 logo are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

API. Application programming interface.

applet. A Java program included in an HTML page. Applets work with Java-enabled browsers, such as Netscape, and are loaded when the HTML page is loaded.

application programming interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program. Net.Data supports the following proprietary Web server APIs for improved performance over CGI processes: ICAPI, GWAPI, ISAPI, and NSAPI.

BLOB. Binary large object.

cache. A type of memory that contains recently accessed data, designed to speed up subsequent access to the same data. The cache is often used to hold a local copy of frequently-used data that is accessible over a network.

caching. The processes of storing frequently-used results from a request to the Web server locally for quick retrieval, until it is time to refresh the information.

Cache Manager. The program that manages a cache for one machine. It can manage multiple caches.

CGI. Common Gateway Interface.

cliette. A long-running process that serves requests from the Web server. The Connection Manager schedules cliette processes to serve these requests.

CLOB. Character large object.

Common Gateway Interface. A standardized way for a Web server to pass control to an application program and receive data back.

Connection Manager. An executable file, `dtwcm`, in Net.Data that is needed to support Live Connection.

cookie. A packet of information sent by an HTTP server to a Web browser and then sent back by the browser each time it accesses that server. Cookies can contain any arbitrary information the server chooses and are used to maintain state between otherwise stateless HTTP transactions. *Free Online Dictionary of Computing*

database. A collection of tables, or a collection of table spaces and index spaces.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

data type. An attribute of columns and literals.

DBMS. Database management system.

firewall. A computer with software that guards an internal network from unauthorized external access.

flat file interface. A set of Net.Data built-in functions that let you read and write data from plain-text files.

HTML. Hypertext markup language.

HTTP. Hypertext transfer protocol.

hypertext markup language. A tag language used to write Web documents.

hypertext transfer protocol. The communication protocol used between a Web server and browser.

ICAPI. Internet Connection API.

ICS. Internet Connection Server.

ICSS. Internet Connection Secure Server.

Internet. An international public TCP/IP computer network.

Internet Connection Server. IBM's unsecure Web server.

Internet Connection Secure Server. IBM's secure Web server.

Intranet. A TCP/IP network inside a company firewall.

ISAPI. Microsoft's Internet Server API.

Java. An operating system-independent object-oriented programming language especially useful for Internet applications.

language environment. A module that provides access from a Net.Data macro to an external data source such as DB2 or a programming language such as Perl. Some language environments are supplied with Net.Data such as REXX, Perl, and Oracle. You can also create your own language environments.

Live Connection. A Net.Data configuration that works with the Connection Manager and Web server API. Live Connection enables database connections to be reused.

LOB. Large object.

middleware. Software that mediates between an application program and a network. It manages the interaction between disparate applications across the heterogeneous computing operating systems. *Free Online Dictionary of Computing*

NSAPI. Netscape API.

null. A special value that indicates the absence of information.

path. A search route used to locate files.

Perl. An interpreted programming language.

port. A 16-bit number used to communicate between TCP/IP and a higher-level protocol or application.

TCP/IP. Transmission Control Protocol / Internet Protocol.

Transmission Control Protocol / Internet Protocol. A set of communication protocols that support peer-to-peer connectivity functions for both local and wide-area networks.

URL. Uniform resource locator.

uniform resource locator. An address that names a HTTP server and optionally a directory and file name, for example: `http://www.software.ibm.com/data/net.data/index.html`.

Web server. A computer running http server software, such as Internet Connection.

Index

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.