IBM Net.Data for OS/400

Net.Data Reference

Version 5

IBM Net.Data for OS/400

Net.Data Reference

Version 5

Note

Before using this information and the product it supports, read the information in "Notices" on page 299.

October 2001 Edition

This edition applies to:

- IBM HTTP Server for AS/400® (Program 5769-DG1), Version 4 Release 4 Modification 0
- IBM HTTP Server for iSeries® (Program 5722-DG1), Version 5 Release 1 Modification 0

and to all subsequent versions and releases until otherwise indicated in new editions.

Preface

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

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® and databases that can be accessed through DRDA®, and using applications written in programming languages such as Java, JavaScript, C, C++, COBOL, and REXX.

Net.Data is a macro processor that executes as middleware on a Web server machine. 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, other data sources, 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 Navigator 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 applications written in, but not limited to, C, C++, RPG, COBOL, Java, or REXX programming languages
- Access databases such as DB2
- Access other data sources such as flat files

Net.Data passes this Web page to the Web server, which in turn forwards the page over the network for display at the browser.

Net.Data can be used in server environments that are configured to use interfaces such as HyperText Transfer Protocol (HTTP) and Common Gateway Interface (CGI). HTTP is an industry-standard interface for interaction between a browser and Web server, and CGI is an industry-standard interface for Web server invocation of gateway applications like Net.Data. These interfaces allow you to select your favorite browser or Web server for use with Net.Data.

About this book

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

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

More information including sample Net.Data macros, demos, and the latest copy of this book, is available from the following World Wide Web site: http://www.ibm.com/systems/i/software/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, variables, 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*.

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 require additional code to work.

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.

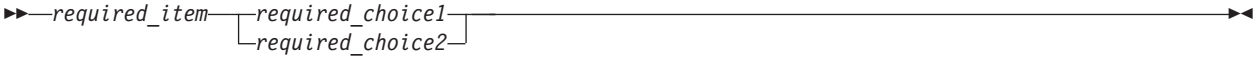
  Diagrams of syntactical units other than complete statements start with the ▸— symbol and end with the —▸ symbol.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.

  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.

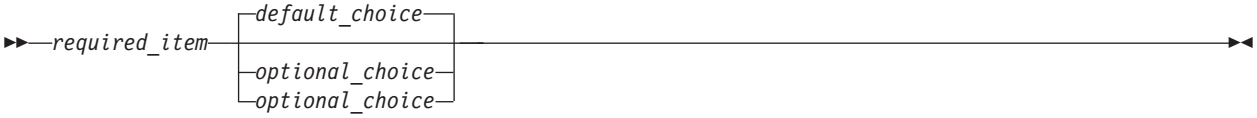
- If you can choose from two or more items, they appear vertically, in a stack.

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



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

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



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

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

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.

Contents

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. 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.

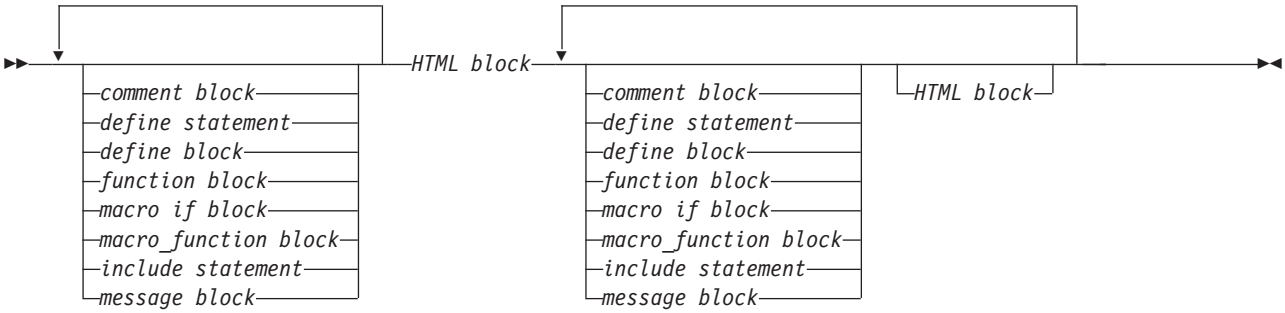
This chapter describes:

Net.Data macro syntax

A Net.Data macro is a 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 or that Net.Data passes to language environments for processing

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 "Macro language constructs" on page 6 for detailed syntax of each element in the global structure.



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

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

Figure 1 on page 2 shows the declaration and presentation parts of the macro.

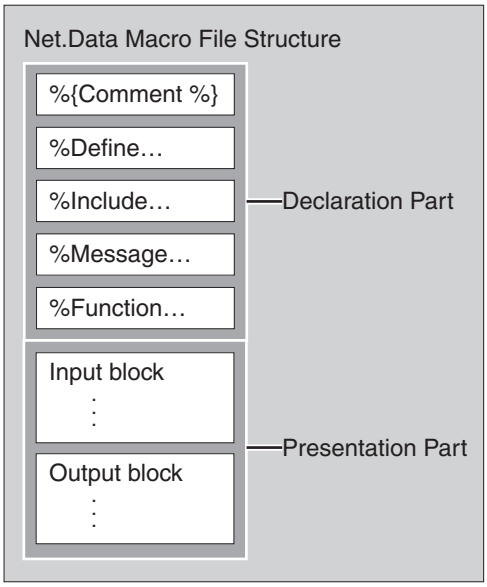


Figure 1. Sample macro structure

Variables and functions that are used in the declaration or generation 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. The declaration part contains the DEFINE and FUNCTION definition blocks. The HTML blocks act as entry points.

```
%{ **********************       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:<br />
<input name="input_data" type="text" size="30" /><br />
<input type="submit" value="enter" /><br />
<hr />
<p>[<a href="/">Home page </a>] </p>
</form>
</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>]</p>
</body></HTML>
%}
```

Figure 2. The Macro 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 B, “Deprecated features,” on page 289, 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 macros. You can find more examples of Net.Data macros at http://www.ibm.com/systems/i/software/netdata/samples/sample.html.

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

A variable name identifies a variable. A variable is an object whose value can change during the execution of a macro.

Variable names must begin with a letter or underscore (_) and contain any alphanumeric characters, underscores, hash marks (#), or periods (.). All variable names are case sensitive except V_*columnName* (See "Net.Data table processing variables" on page 61 for more information about these two exceptions.).

Variable reference

Variable references returns the value of a variable and is specified with $ and (). For example: if VAR = 'front', $(VAR) returns the value 'front'. 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.

You can dynamically generate a variable reference by including variable references, strings, and function calls within a variable reference. For example: if frontside = 'blue', $($(VAR)side) returns the value 'blue'. If you reference a dynamically-generated variable that does not follow the variable name rules, Net.Data resolves the reference to an empty string.

Restrictions:

- Net.Data strings are terminated by NULL characters (binary zeroes). If your data source returns data containing the NULL character, Net.Data will terminate the string stored in the variable at that point in the data stream.
- Variable references cannot be used as an OUT parameter to a function call.
- Leading and trailing whitespace is ignored.
- Whitespace (including a newline character) is not allowed between function calls, strings, and variable references.
- A variable reference returns an empty string if any whitespace exists within the name of the variable.

Syntax:

Notes:

1 String can contain only the characters that are allowed in variable names: alphanumeric characters, underscores (_), hash marks (#), or periods (.).

Example 1: Variable reference

If you have defined a variable `homeURL`:

```
%DEFINE homeURL="http://www.ibm.com/"
```

You can refer to the homepage as $(homeURL) and create a link:

```
<a href="$(homeURL)">Home page</a>
```

Example 2: Dynamically-generated variable reference

You can dynamically generate variable references that in turn dynamically reference a field value in a row:

```
%define{
var1="value1"
var2="value2"
var3="value3"
@DTW_ASSIGN (INDEX, "1")
%}
%WHILE (INDEX < 3) {
 $(var$(INDEX))
 @DTW_ADD(INDEX, "1", INDEX)
 %}
```

Returns:

```
value1
value2
value3
```

Example 3: A dynamic variable reference with nested variable references and a function call

```
%define my = "my"
%define u = "lower"
%define myLOWERvar = "hey"


$($(my)@dtw_ruppercase(u)var)
```

The variable reference returns the value of `hey`.

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.

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 newline character. For example,

```
%define multiline = {
first line
second line
%}
```

To specify double quotes inside a quoted string, use two pairs of double quotes. A string used as function argument or as term in a comparison expression can contain double quotes. For example, if you define a string value as:

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

The value of *result* is:

```
"Hello world!"
```

A presentation statement is a string.

Strings used as function arguments, terms, and variable values can contain variable references and function calls. In the following example, the function call `myfunc2` has a string parameter that contains a variable reference and a function call.

```
%HTML(report) {
  @myfunc2("abc$(var1)@myfunc()")
%}
```

Net.Data resolves the variable reference `$(var1)` and the function call `@myfunc()`, rather than interpreting them literally as part of the string, before passing the string to the function `myfunc2`.

Macro language constructs

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

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

Purpose

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

The COMMENT block can also be used in the Net.Data initialization file.

Syntax

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 or the Net.Data initialization file

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          %}
%}
```

Example 5: Comments in the Net.Data initialization file

```
...
%{ restrict for general use %}
 DTW_DIRECT_REQUEST no
...
```

DEFINE block or statement

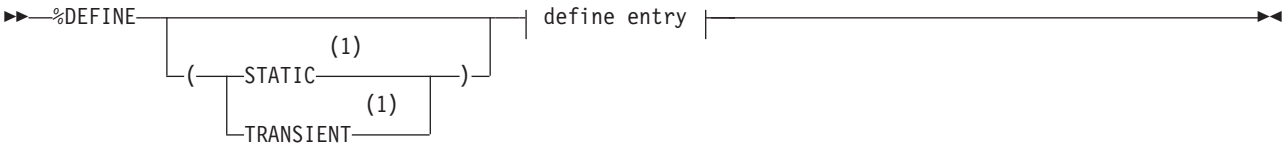
Purpose

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

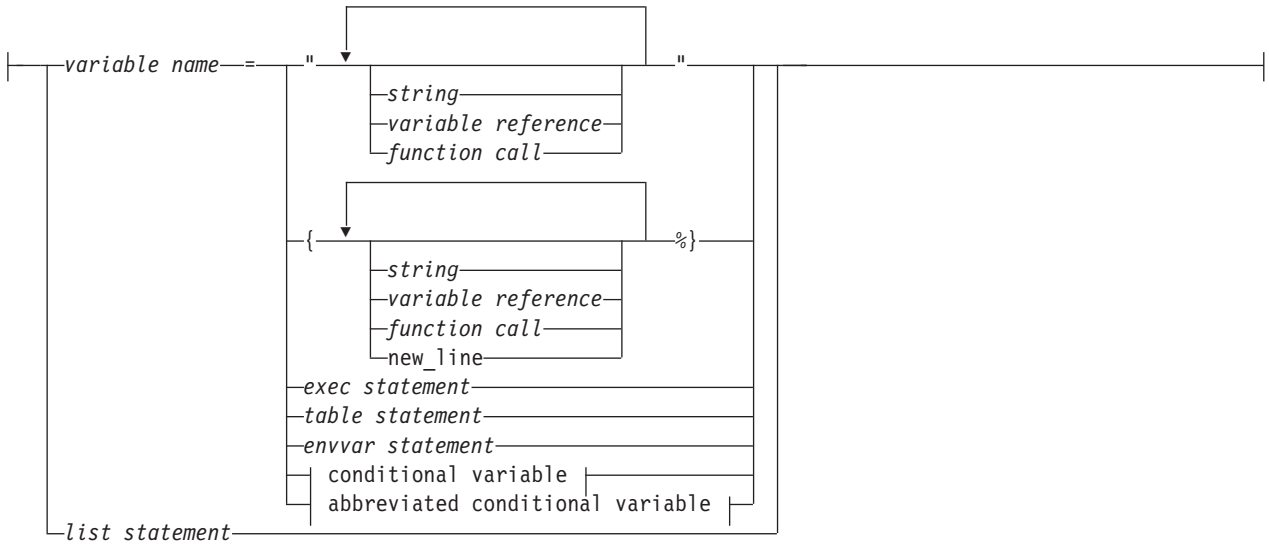
The variable definition can be on a single line, using double quotes (""), or can span multiple lines, using brackets and a percent sign ({ %}). After the variable is defined, you can reference it anywhere in the macro.

Syntax



define entry

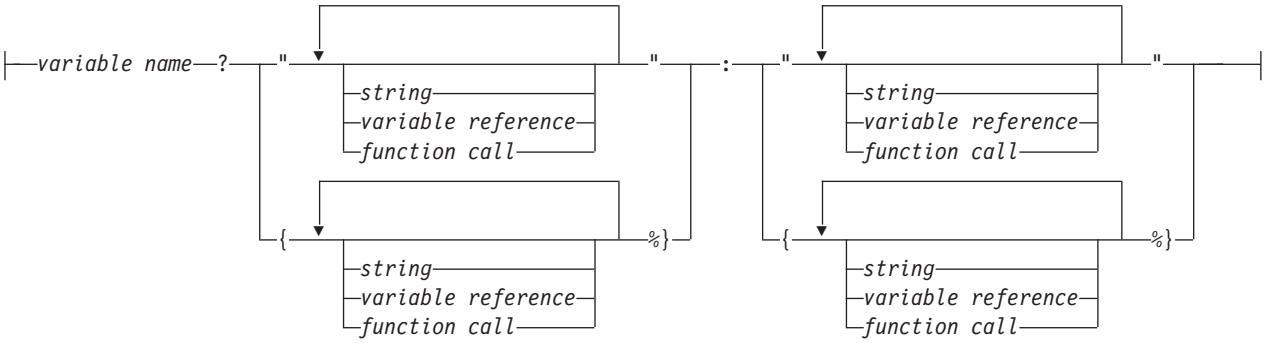
define entry:



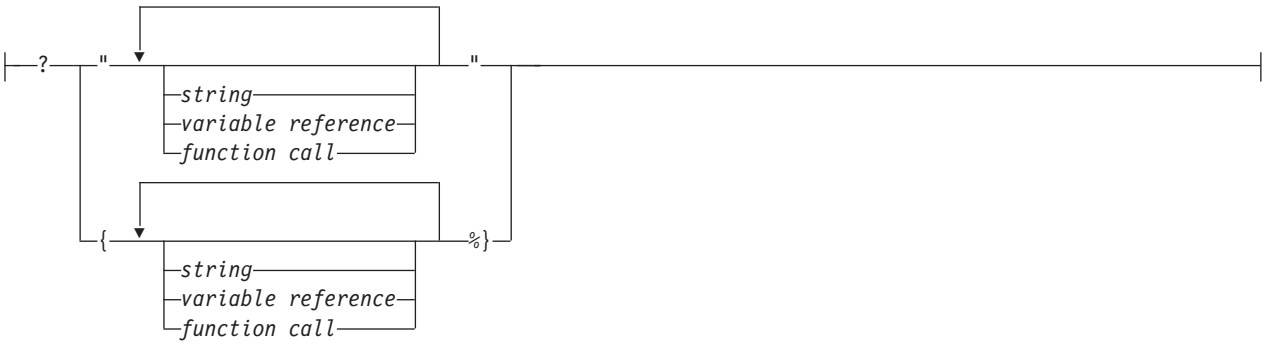
conditional variable

abbreviated conditional variable

conditional variable:



abbreviated conditional variable:



Notes:

1 STATIC and TRANSIENT are keywords for persistent macros.

Values

%DEFINE
A keyword that defines variables.

STATIC
A keyword that specifies that the variable retains its value across macro invocations within a persistent transaction. This is the default for persistent macros.

TRANSIENT
A keyword that specifies that this variable does not retain its value across macro invocations. This is the default for non-persistent macros.

define entry:

variable name
A name that identifies a variable. 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 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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function call (@)” on page 21 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 49 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 whether another variable or string is empty.

abbreviated conditional variable
Sets the value of a variable based on whether another variable or string is empty. 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 34 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
- 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* contains the empty string.

Example 3: Definition of multiple variables

```
%DEFINE{  DATABASE = "testdb"
          home = "http://www.ibm.com/software"
          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.

```
%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.

Syntax

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.

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 Net.Data encounters an executable variable in a macro, 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. If it locates the executable program, Net.Data runs the program.

Authorization Tip: Ensure that the user ID under which Net.Data executes 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 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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See "Function call (@)" on page 21 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

The following Net.Data-provided language environments support the EXEC statement:

- REXX
- System
- DIRECTCALL

Examples

Example 1: Executable file referenced by a variable

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

%HTML (Report) {
<p>Here is the report you requested:</p>
<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 %}
%}
```

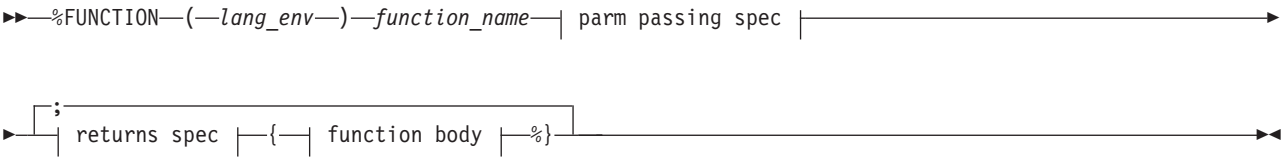
This example executes `mypgm.cmd` when the function `my_rexx_pgm` is called.

FUNCTION block

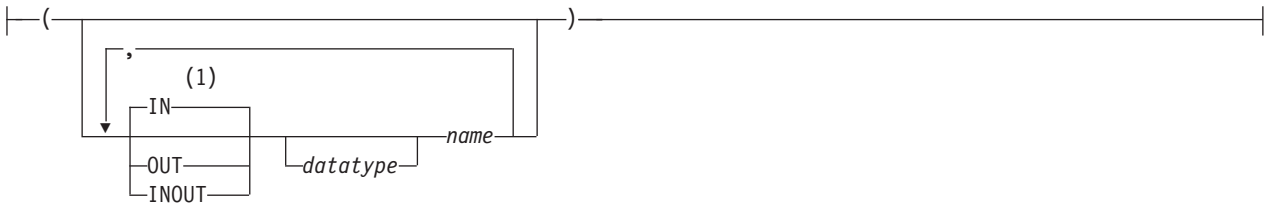
Purpose

Defines a subroutine that Net.Data invokes from the macro. 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.

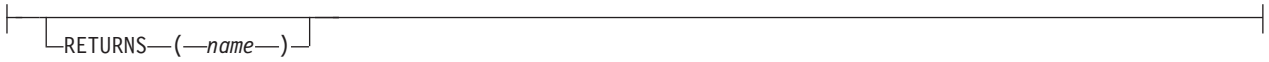
Syntax



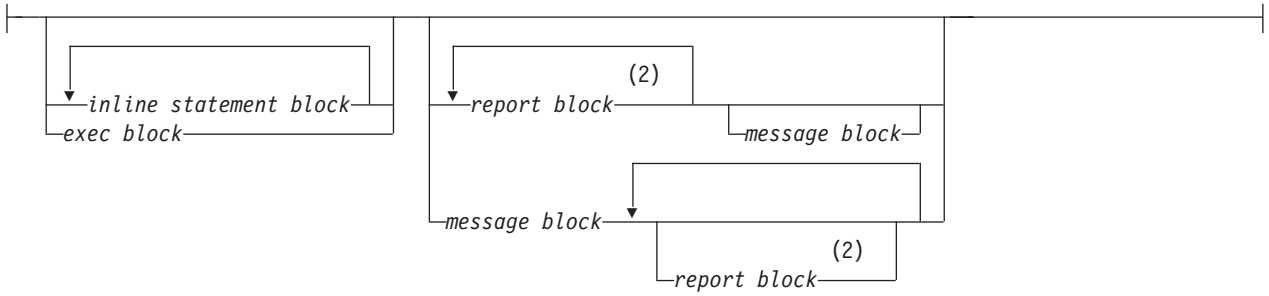
parm passing spec:



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 valid for the SQL language environmentwhen processing stored procedures that return result sets and functions calling any language environment.

Values

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

lang_env
The language environment that processes the function body. See the *Net.Data Administration and Programming Guide* 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, underscore, or period 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.

datatype
Specifies the datatype of the parameter. For a list of supported datatypes for stored procedures, see the *Net.Data Administration and Programming Guide*.

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

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, or SQL. See the *Net.Data Administration and Programming Guide* 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).

When characters that match Net.Data language constructs syntax are used in the language statements section of a function block as part of syntactically valid embedded program code (such as REXX), they can be misinterpreted as Net.Data language constructs, causing errors or unpredictable results in a macro.

Use one of the following methods to use Net.Data special characters as part of your embedded program code, without having them interpreted by Net.Data as special characters:

- Use the EXEC statement to call the program code, rather than putting the code inline.
- Use a variable reference to specify the special characters.

exec block
The EXEC block. The name of an external program that executes when the function is called.

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 path configuration statement in the initialization file. The resulting string is passed to the language environment to be executed.

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 44 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 40 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
- The length of SQL statements in the inline statement block should not exceed your database limits. Refer to your database documentation to determine what the limit is.

Examples

The following examples are general and do not cover all language environments. See the *Net.Data Administration and Programming Guide* 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>
 <p> @my_rexx_pgm(w, x, y, z) </p>
 <p> Modified variable values: $(w) $(x) $(z) </p>
%}
```

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

```
%DEFINE customer_name="IBM"
%DEFINE customer_order="12345"

%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>
     <p>$(NLIST) </p>
     <ul>
     %ROW{
       <li>$(V1)  $(V2)  $(V3)  $(V4) </li>
     %}
     </ul>
        %}
%}
```

```
%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. The NLIST and Vn 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 C program

- Net.Data macro:

```
%FUNCTION(DTW_SYSTEM) hello(IN name) RETURNS(result) {
  %exec{  /QSYS.LIB/NETDATA.LIB/hello.PGM %}
%}
%HTML(INPUT) {
  @hello("john")
%}
```

- C program `hello.c`:

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

int main()
{
    char env_buffer[100];
    sprintf(env_buffer, "result=Hello %s", getenv(name));
    putenv(env_buffer);

    return 0;
}
```

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.
- Similar to input parameters, output parameters must be passed back to the language environment as system environment variables.

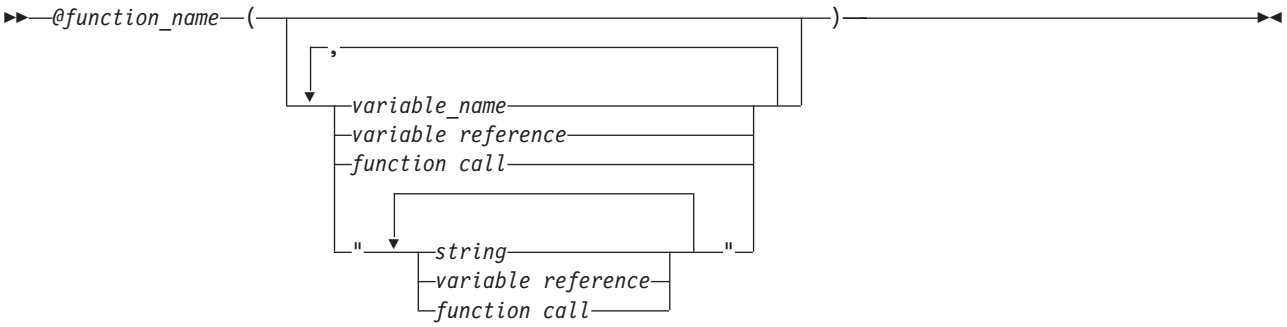
In the example above, when the `@hello` function call is encountered, Net.Data performs variable substitution on the executable statements. In this example, there are no Net.Data variables in the executable statements, so no variable substitution is performed. The executable statements and parameters are passed to the System language environment, which performs a putenv() on all input variables. In the example above, the system environment variable `name` is created with a value of `john`. The C program then retrieves the `name` environment variable by using the getenv() API, and then generates an environment variable named `result` with the value set to "Hello john". When the C program ends, the `@hello("john")` function call results in "Hello john" being returned to the Net.Data client.

Function call (@)

Purpose

Invokes a 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, underscore, or period characters.

variable name
A name that identifies a variable. 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 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 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
- WHILE block
- MESSAGE block
- MACRO_FUNCTION 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
- OUT or INOUT parameter values cannot contain variable references, function calls, or literal strings.

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?</p>
<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" />
</form>
%}

%HTML (Report){
<p>Here are the columns you selected:</p>
<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>
 <p> @my_rexx_pgm(w, x, y, z) </p>
 <p> Modified variable values: $(w) $(x) $(z) </p>
%}
```

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)</p>
%}
```

Example 4: A macro that illustrates the use of the INOUT parameter.

```
%DEFINE a = "initial value of a"

%FUNCTION(DTW_REXX) func1(INOUT x) {
  Say 'value at start of function:<br />
  Say 'x =' x
  Say '<p>'
  x = "new value of a"
```

```
  Say '<p>'
  %REPORT {
    <p>value at start of report block:<p>
    x = $(x)<br />
    @dtw_assign(x, "newest value of a")
    value at end of report block:<br />
    x = $(x)<br />
  %}
%}

%HTML (Report) {
  initial values:<br />
  a = $(a)<br />
  @func1(a)
  value after function call:<br />
  a = $(a)<br />
%}
```

Resulting output:

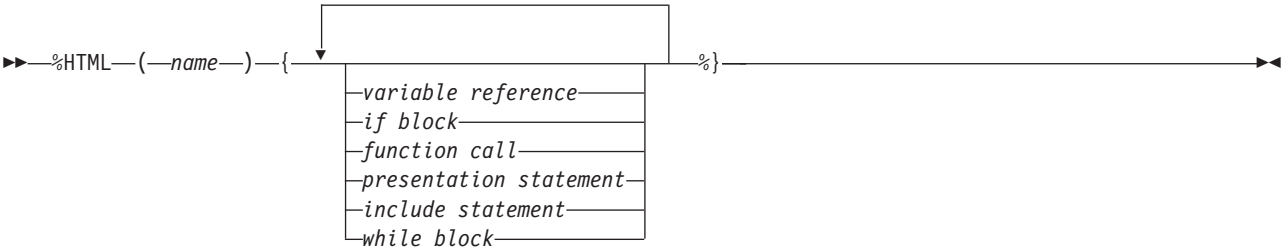
```
initial values:
a = initial value of a

value at start of function:
x = initial value of a

value at start of report block:
x = new value of a

value at end of report block:
x = newest value of a

value after function call:
a = newest value of a
```

HTML block

Purpose

The HTML block is an entry point into the macro. The name of the HTML block to be executed is specified on the URL when Net.Data is invoked. The HTML block can contain most Net.Data macro language statements and any valid presentation statements, such as HTML, XML, and Javascript.

Syntax



Values

%HTML
The keyword that specifies that the block is a presentation block.

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

variable reference
Returns the value of a 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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See "Function call (@)" on page 21 for syntax and examples.

presentation statements
Includes any text, such as HTML tags, to be consumed by the client.

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.

while block
The WHILE block. Performs looping with conditional string processing. See "WHILE block" on page 51 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
- IF block
- presentation statements
- INCLUDE statement
- WHILE block
- Variable references
- Function calls

Examples

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

```
%HTML(my.report){
%INCLUDE "header.html"
<p>You can put <em>any</em> HTML in an HTML block.
An SQL function call is made like this:</p>
@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. 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.

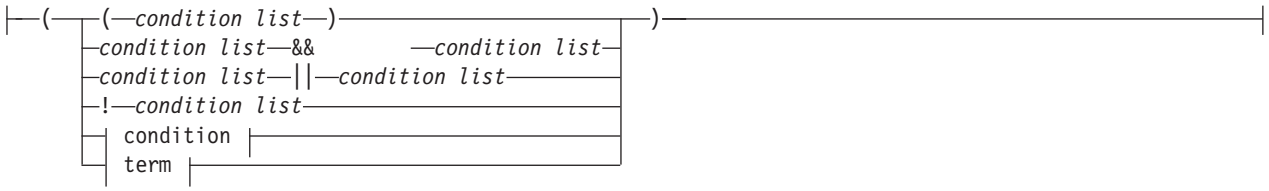
In the following example, the nested IF block must follow the rules used when it is inside an HTML block.

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

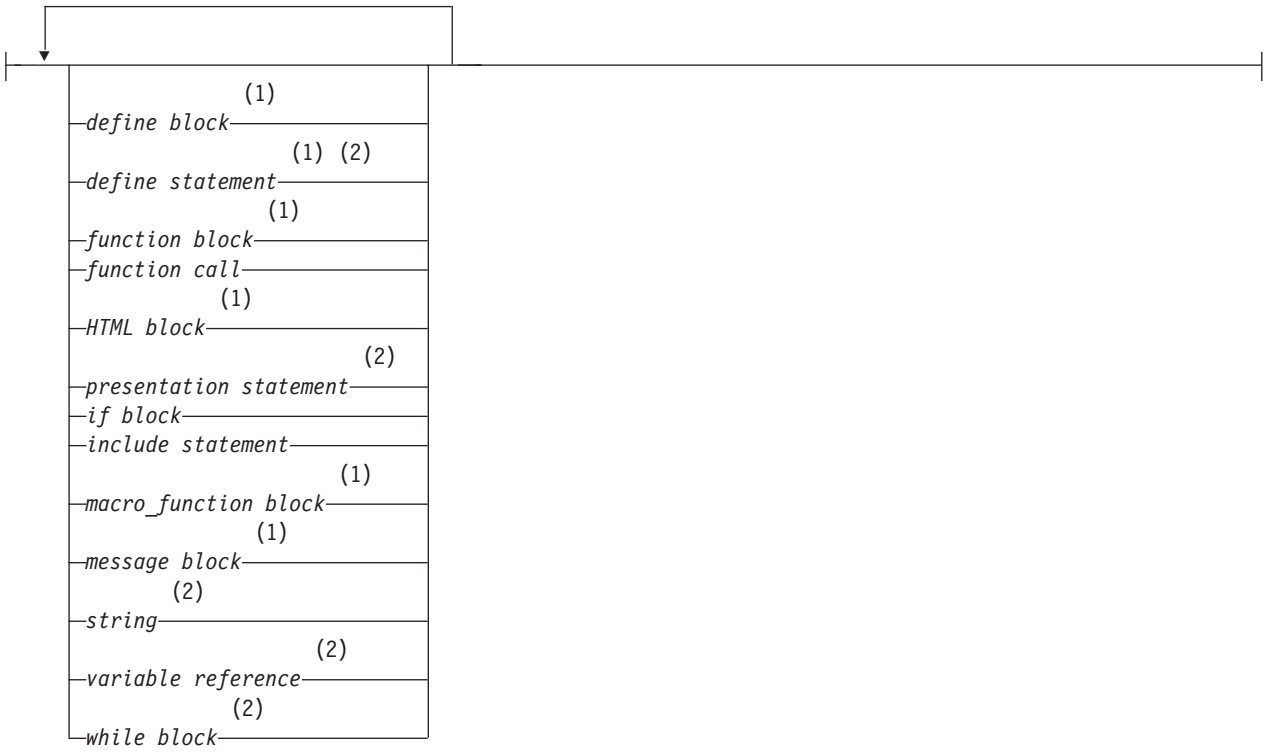
You can nest up to 1024 IF blocks.

Syntax

condition list:



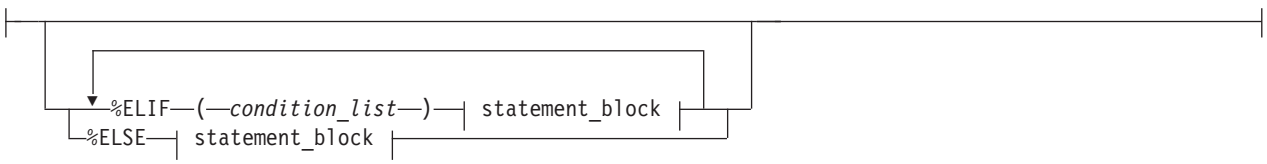
statement_block:



condition:

term:

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, ROW 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.

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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See "Function call (@)" on page 21 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.

presentation statement
Includes any textual data, such as HTML or XML tags, to be returned to the client.

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.

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 36 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 40 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 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 51 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
- 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
- Function calls
- IF block
- INCLUDE statement
- presentation statement
- String
- Variable reference
- WHILE block

You can nest up to 1024 IF blocks.

Examples

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

```
%DEFINE a = "1"
%DEFINE b = "2"
...
%IF (OUT_FORMAT = "HTML")
   %define DTW_HTML_TABLE = "YES"
%ELSE
   %define DTW_HTML_TABLE = "NO"
%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...</p>
%ELIF (ROW_NUM == "100")
    <p>The table is now full...</p>
%ELSE
    <p>The table has overflowed...</p>
%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. 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 user ID under which Net.Data executes has access rights to any files referenced by any INCLUDE statements. 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.

Syntax

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, underscore, or period characters.

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

variable reference
Returns the value of a 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

Function calls in the string are not allowed.

You can nest up to ten INCLUDE statements.

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

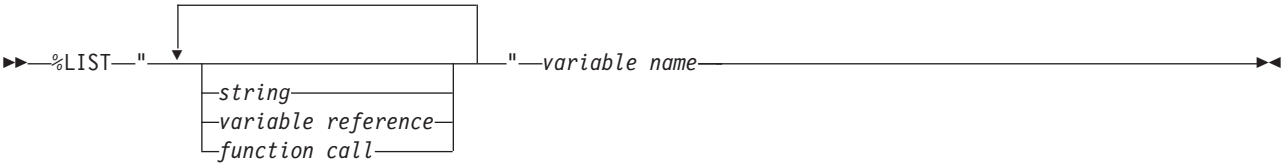
```
%define REMOTE_USER = %ENVVAR
%include "$(REMOTE_USER).hti"
```

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



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 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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See "Function call (@)" on page 21 for syntax and examples.

variable name
A name that identifies a variable. 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
conditions="cond1='Sao Paolo'"
```

```
conditions="cond2='Seattle'"
conditions="cond3='Shanghai'"
whereClause=conditions ? "WHERE $(conditions)" : ""
%}
```

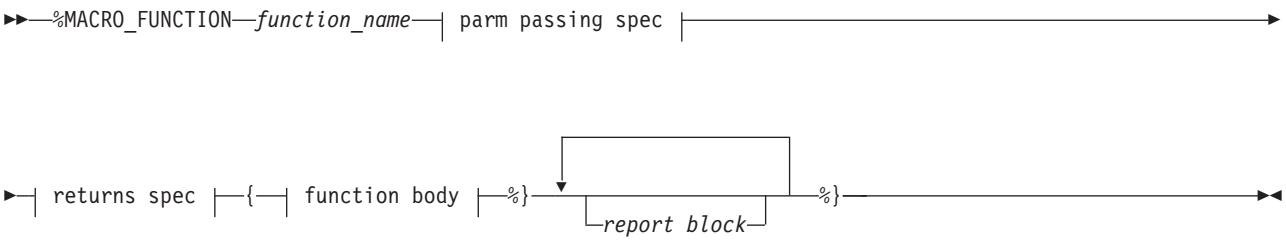
For more information on using LIST statements with variables, see “List variables” on page 58.

MACRO_FUNCTION block

Purpose

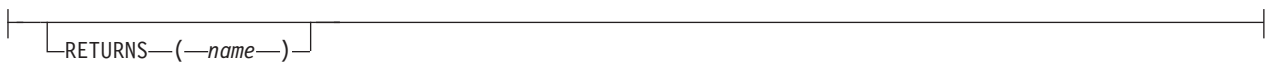
Defines a subroutine that can be invoked from the Net.Data macro. The executable statements in a MACRO_FUNCTION block must be Net.Data macro language source statements.

Syntax



parm passing spec:

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.

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, underscore, or period 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. *name* can represent a Net.Data table or a result set.

returns spec:

RETURNS
Declares the variable that contains the function value after the function completes.

function body:

variable reference
Returns the value of a 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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See “Function call (@)” on page 21 for syntax and examples.

presentation 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.

while block
The WHILE block. Performs looping with conditional string processing. See “WHILE block” on page 51 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 44 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

The MACRO_FUNCTION block can contain these elements:

- Comment block
- presentation statements
- IF block
- INCLUDE statement
- REPORT block
- 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
%}
```

Example 3: A macro function that contains a REPORT block

```
%MACRO_FUNCTION myfunc (INOUT table) {
   %REPORT {
      <table>
      %ROW {
         <tr><td>$(V1)</td><td>$(V2)</td></tr>
      %}
      </table>
   %}
%}
```

Example 4: A macro function that uses the RETURNS keyword

```
%MACRO_FUNCTION myfunc () RETURNS(VALUE)  {
   @DTW_ASSIGN(VALUE, "Success...")
%}
```

MESSAGE block

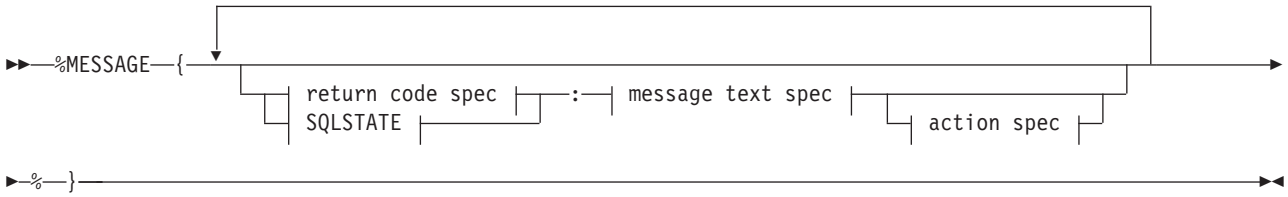
Purpose

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.

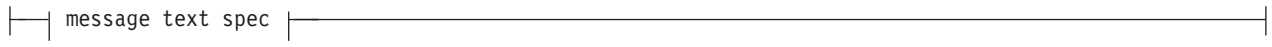
Syntax



return code spec

SQLSTATE

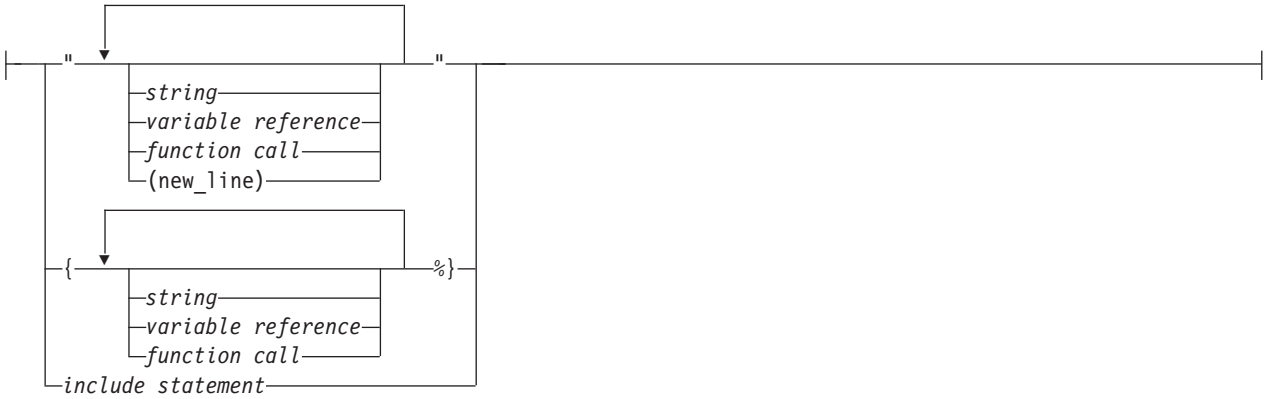
message text spec



return code spec:

SQLSTATE:

message text spec:



action spec:

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. This value is matched with the Net.Data variable SQL_STATE.

state_id
The SQLSTATE. An alphamumeric string of five characters (bytes) with a format of *ccsss*, where *cc* indicates class and *sss* indicates subclass.

message text spec
A string that is sent to the Web browser if either the RETURN_CODE matches the *return_code* value or the SQL_STATE variable matches the SQLSTATE 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 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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See "Function call (@)" on page 21 for syntax and examples.

action spec
Specifies what action Net.Data takes if the RETURN_CODE variable matches the *return_code* value, or the SQL_STATE variable matches the SQLSTATE 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
- presentation statements
- Strings
- INCLUDE statement

Examples

Example 1: A local MESSAGE block

```
%{ local message block inside a FUNCTION block %}
%FUNCTION(DTW_REXX) my_function() {
  %EXEC { my_command.cmd %}
 %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>"%}
      : exit
  %}
```

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.

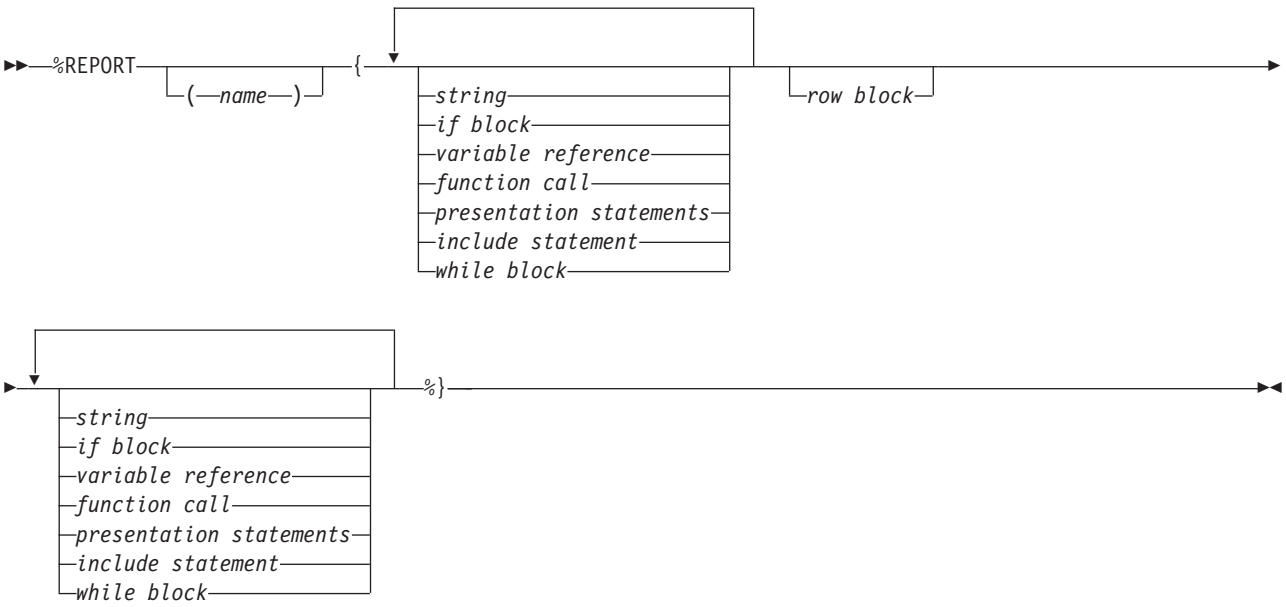
```
 %message {
   %include "rc1000.msg"
   %include "rc2000.msg"
   %include "defaults.msg"
 %}
```

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
This value represents a Net.Data table or result set. See the Report Block section in the *Net.Data Administration & Programming Guide* for more information.

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 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 FUNCTION or MACRO_FUNCTION blocks, or a Net.Data built-in function with specified arguments. See "Function call (@)" on page 21 for syntax and examples.

presentation 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.

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 47 for syntax and examples.

while block
The WHILE block. Performs looping with conditional string processing. See "WHILE block" on page 51 for syntax and examples.

Context

The REPORT block can be found in these contexts:

- FUNCTION statement or block
- MACRO_FUNCTION block

Restrictions

The REPORT block can contain these elements:

- Comment block
- IF block
- INCLUDE statements
- ROW blocks
- WHILE blocks
- Function calls
- presentation statements
- Strings
- Variable references

Examples

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

```
%FUNCTION(DTW_SQL) mytable() {
 %REPORT{
 <h2>Query Results</h2>
 <p>Select a name for details.</p>
 <table border="1">
 <tr><td>Name</td><td>Location</td></tr>
 %ROW{
 <tr>
 <td>
 <a href="/cgi-bin/db2www/name.mac/details?name=$(V1)&loc=$(V2)">$(V1)
 </a></td>
 <td>$(V2)</td>
 </tr>
 %}
</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

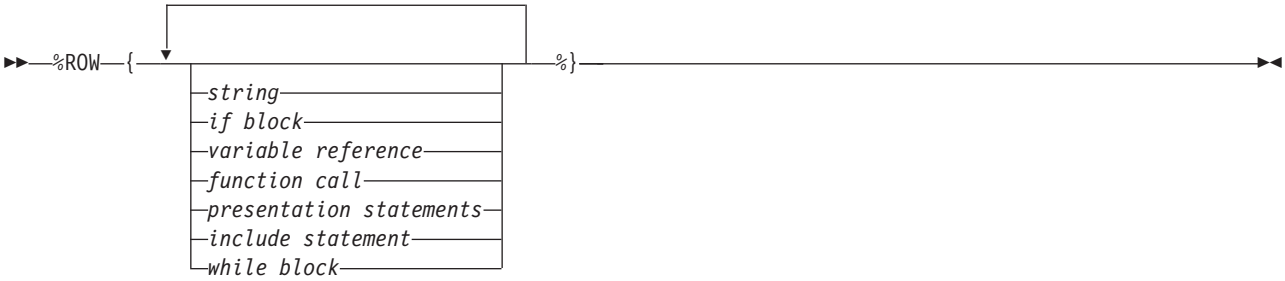
additional details about the name.

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 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 FUNCTION or MACRO_FUNCTION blocks, or built-in functions with specified arguments. See "Function call (@)" on page 21 for syntax and examples.

presentation 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.

while block
The WHILE block. Performs looping with conditional string processing. See "WHILE block" on page 51 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
- WHILE blocks
- Function calls
- Variable references
- presentation 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.</p>
<table border="1">
<tr><th>Name</th><th>Location</th></tr>

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

</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. The variable contains a set of rows and columns including a row of column headers describing the fields in each row. A table statement can only be in a DEFINE statement or block.

When a TABLE variable is referenced while executing an HTML block, Net.Data displays the content of the table as either a plain character table or, if the DTW_HTML_TABLE variable is set to YES, as an HTML table.

Syntax

upper limit:

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. If the upper limit value is not specified, the table can contain an unlimited number of rows.

number
A string of digits. 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

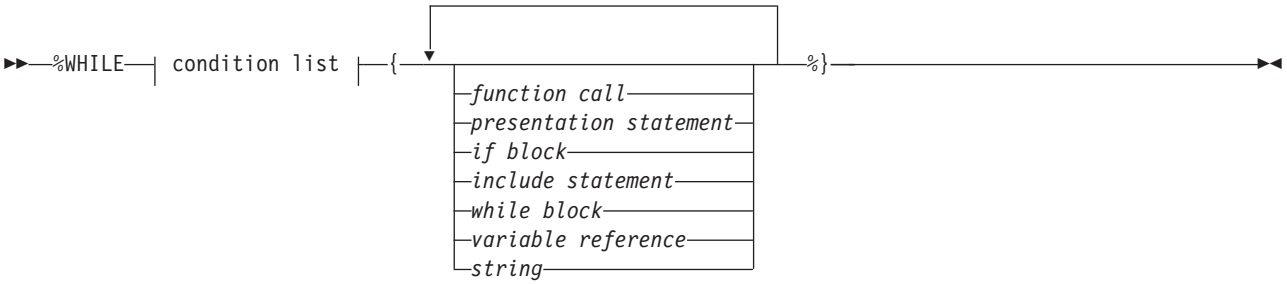
```
%DEFINE myTable3=%TABLE(ALL)
```

WHILE block

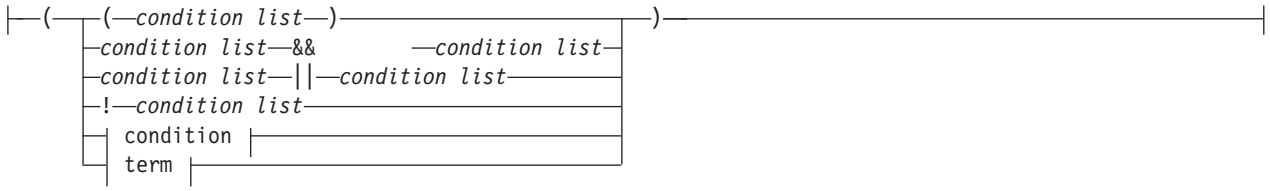
Purpose

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.

function call
Invokes one or more FUNCTION or MACRO_FUNCTION blocks, or built-in functions with specified arguments. See "Function call (@)" on page 21 for syntax and examples.

presentation 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.

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.

while block
The WHILE block. Performs looping with conditional string processing.

variable reference
Returns the value of a 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
A name that identifies a variable. 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
- IF block
- WHILE block
- Strings
- presentation 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) {
%{ generate table tag and column headings %}
<table border="1">
<tr>
<th>Item #</th>
<th>Description</th>
</tr>
%WHILE (loopCounter <= "100") {
  %{ generate individual rows %}
  <tr>
  <td>$(loopCounter)</td>
  <td>@getDescription(loopCounter)</td>
  </tr>

  %{ increment loop counter %}
  @dtw_add(loopCounter, "1", loopCounter)
%}
</table>
%}
```

Chapter 2. Variables

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

"User-defined variables"

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

- **"Conditional variables" on page 56**

  Assign a variable value based on the value of another variable or string.
- **"Environment variables" on page 57**

  Use the ENVVAR language construct to reference environment variables.
- **"Executable variables" on page 57**

  Use the EXEC language construct to invoke other programs from a variable reference.
- **"Hidden variables" on page 58**

  Hide variable reference from HTML source.
- **"List variables" on page 58**

  Build a delimited string of values using the LIST language construct.
- **"Table variables" on page 60**

  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 61**

  Defined by Net.Data to let you process Net.Data tables. Use these variables to access data from SQL queries and function calls. They are only recognized inside REPORT or ROW blocks, unless otherwise specified.
- **"Net.Data REPORT block variables" on page 70**

  Help you customize reports from a function. You can define report variables in the DEFINE section, or assign them in any Net.Data block.
- **"Net.Data language environment variables" on page 77**

  Help you customize the way FUNCTION blocks are processed, using language environments.
- **"Net.Data miscellaneous variables" on page 90**

  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.

User-defined variables

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

Conditional variables

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

The syntax to conditionally set a variable is:

```
condVar = testVar ? trueValue : falseValue
```

Where:

condVar
The conditional variable to be set.

testVar
A test variable used to determine the condition. An empty string is evaluated as false.

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

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

Example 1: A conditional variable defined with two possible values

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

`varB` is the test. So, `varA` is assigned either "value_1" or "value_2", depending on whether `varB` exists and is not empty.

Example 2: 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 empty or not defined, the WHERE clause changes to be an empty string. For example, if *prod_inp* is empty, the WHERE clause is:

```
WHERE custid = IBM
```

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

Environment variables

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.ibm.com`.

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

See "ENVVAR statement" on page 13 for more information about the ENVVAR statement.

Executable variables

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, 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. 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"
```

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

If the presentation block contains the following:

```
Today is $(date)
```

The browser displays the results of the executable:

```
Today is 02-14-2001
```

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

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).

Do not reference hidden variables with dynamically constructed variable names.

Example 1: Hidden variables in a HTML form

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

%FUNCTION(DTW_SQL) mySelect() {
  SELECT $(field) FROM customer
%}

%HTML(INPUT) {
<form ...>
<p>Select fields to view:</p>
<select name="field">
<option value="$$(name)"> Name</option>
<option value="$$(addr)"> Address</option>
.
.
</select>
.
.
</form>
%}
```

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. 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

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 34 for syntax and more information.

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

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

%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>
%}

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

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

If no boxes are checked in the HTML form, then *conditions* is empty, so `whereClause` is also empty 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.

Example 3: Function calls

```
%define {
   %list  "$(funct1)" funct
   funct1="@dtw_rgetenv("PATH")"
   funct="@dtw_rgetenv("DB2INSTANCE")"
   funct="@dtw_ruppercase("Test for LIST statement contains Function Calls")"
   funct="@dtw_rlowercase("THIS'S A TEST")"
                               funct="your_function"
  %}
```

```
  %html_report {
  funct = $(funct)
  %}
```

Note to Thuyanh or other developers: I need to include a paragraph describing this example. Please provide the text and I'll edit it!

Table variables

The table variable defines a collection of related data. It contains a set of rows and columns including a row of column headers. 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 or by using table built-in functions. 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 49 for syntax and more information.

When a TABLE variable is referenced, Net.Data displays the content of the table as either a plain character table, or as an HTML table if the DTW_HTML_TABLE variable is set to YES.

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)
</p>
</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.

N*n*

Purpose

The column name returned by a function call or query for column `n`.

You can reference N*n* in REPORT and ROW blocks.

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(OUT col1) {
  select * from atable
  %report {
    @dtw_assign(col1, N1)
    %row{ %}
  %}
%}

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

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 154.

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>$(N3)</td></tr>
%ROW{
<tr><td>$(V1)</td><td>$(V2)</td><td>$(V3)</td></tr>
%}
</table>

%}
```

NLIST

Purpose

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

You can reference NLIST in REPORT and ROW blocks.

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

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.

You can reference NUM_COLUMNS in REPORT and ROW blocks.

Examples

Example 1: NUM_COLUMNS used as a variable reference with NLIST

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

ROW_NUM

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.

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.

You can reference ROW_NUM only from 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></tr>
%ROW{
<tr><td> $(ROW_NUM) </td> <td> $(V_custname) </td></tr>
%}
</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

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.

Language Environment Restriction: Use this variable only with the following database language environments:

- SQL

Required: You must set DTW_SET_TOTAL_ROWS to YES to use this variable. See "DTW_SET_TOTAL_ROWS" on page 83 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)</li>
%}
</ul>
Names found: $(TOTAL_ROWS)
%}
```

V_*columnName*

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.

You can reference V_*columnName* only within a ROW block.

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

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

```
%FUNCTION(DTW_SQL) myQuery() {
 SELECT NAME, ADDRESS from $(qtable)
 %REPORT{

 %ROW{

 Value of NAME column in row $(ROW_NUM) is $(V_NAME).<br />
 %}
 %}
 %}
```

VLIST

Purpose

A list of all the field values for the current row being processed in a ROW block.

You can reference VLIST only within a ROW block. The default separator is a space.

Net.Data assigns the variable for each row in the table; reference the variable to obtain the values of all the fields in the current row. To use this variable outside the block, assign the value of VLIST to a previously defined global variable or an OUT or INOUT function parameter variable.

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)</li>
%}
</ol>
%}
```

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

```
%DEFINE %LIST "<br />" VLIST

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

V*n*

Purpose

The value for the specified column number *n* for the current row.

You can reference V*n* only within a ROW block.

Net.Data assigns the variable for each field in 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></tr>
%ROW{
<tr><td>$(V1)</td>
<td><a href="mailto:$(V2)">$(V2)</a></td>
<td>$(V3)</td></tr>
%}
</table>
%}
```

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

Net.Data REPORT block variables

These variables help you customize your reports. Each variable has a default value. You can override the default value by assigning a new value to the variable.

ALIGN

Purpose

Controls leading and trailing spaces used with the table processing variables NLIST and VLIST.

Performance Tip: Use ALIGN only when necessary as it requires that Net.Data determine the maximum column length for all columns in the table to calculate padding requirements. This process can impact performance.

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)</p>
```

DTW_DEFAULT_REPORT

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"|"MULTIPLE"
```

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.
MULTIPLE	Net.Data generates default reports for result sets or output tables that are not assigned to a REPORT block, in functions with multiple REPORT blocks

Examples

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

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

DTW_HTML_TABLE

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

Purpose

Specifies the number of rows in a table that are processed in a function REPORT block or during the generation of a default report if a REPORT block is not specified.

The database language environments use this variable to limit the number of rows returned, which can subtantially improve performance for large result sets. Use this variable with START_ROW_NUM to break queries with large result sets into smaller tables, each on its own HTML page.

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

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

Purpose

Specifies the starting row number in a table that will get processed in a function REPORT block, during the generation of a default report if a REPORT block is not specified, or the setting of a Net.Data table variable.

The database language environments use this variable to determine the starting row in the result set to begin processing. To substantially improve performance for large result sets, use this variable with RPT_MAX_ROWS to break queries with large result sets into smaller tables.

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. The default value is 1. 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.

Examples

Example 1: Scrolling with HTML form Next and Previous buttons

```
%define START_ROW_NUM = "1"
%define RPT_MAX_ROWS = "13"
%define DTW_SET_TOTAL_ROWS = "yes"
%function(DTW_SQL) select() {
  select * from usrnd01.customer
  %REPORT {
    @DTW_ADD(START_ROW_NUM, RPT_MAX_ROWS, next_row_num)
    @DTW_SUBTRACT(START_ROW_NUM, RPT_MAX_ROWS, prev_row_num)
    @DTW_SUBTRACT(next_row_num, "1", last_row)
    <h2>Reporting rows $(START_ROW_NUM) through $(last_row)</h2>
    <table BORDER="2">
    <tr><th>$(N1)</th><th>$(N2)</th><th>$(N3)</th>
        <th>$(N4)</th><th>$(N5)</th></tr>
    %ROW {
    <tr><td>$(V1)</td><td>$(V2)</td><td>$(V3)</td>
        <td>$(V4)</td><td>$(V5)</td></tr>
    %}
    </table>
    <p> </p>
    <p><h3>
    %IF (START_ROW_NUM > RPT_MAX_ROWS)
      <a href="report?START_ROW_NUM=$(prev_row_num)">PREVIOUS</a> |||
    %ELSE
    PREVIOUS |||
    %ENDIF
    %IF (next_row_num < TOTAL_ROWS)
      <a href="report?START_ROW_NUM=$(next_row_num)">NEXT</a>
    %ELSE
    NEXT
    %ENDIF
    </h3>
```

```
    TOTAL_ROWS = $(TOTAL_ROWS)</p>
  %}
%}
%html(report) {
<html><body>
@select()
</body></html>
%}
```

Net.Data language environment variables

Use these variables with functions to help you customize the way FUNCTION blocks are processed by language environments. Each variable has a default value. You can override the default value by assigning a new value to the variable.

DATABASE

Purpose

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

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

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.

DTW_APPLET_ALTTEXT

Purpose

Displays HTML tags and text to browsers that do not recognize the APPLET tag and is used with the the Applet built-in functions.

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

Values

DTW_APPLET_ALTTEXT="*text_and_markup*"

Table 8. DTW_APPLET_ALTTEXT Values

Values	Description
text_and_markup	Text and markup that should be displayed to 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.</p>"
```

DTW_EDIT_CODES

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, the corresponding column in the resulting table 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 9. 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_PAD_PGM_PARMS

Purpose

Indicates to a language environment whether character parameters (data type of CHAR or CHARACTER) are to be padded with blanks when they are being passed to a program or stored procedure.

For IN or INOUT parameters, if the length of parameter value is less than the precision that is specified, blanks are inserted to the right of the parameter value until the length of the parameter value is the same as the precision.

For OUT parameters, the parameter value is set to *precision* blanks.

After the call to the program or stored procedure, all trailing blanks are removed from OUT and INOUT parameter values.

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. If DTW_PAD_PGM_PARMS is not defined in the macro, it uses the value in the Net.Data initialization file.

DTW_PAD_PGM_PARMS is supported by the Direct Call and SQL language environments.

Values

```
DTW_PAD_PGM_PARMS="YES"|"NO"
```

Table 10. DTW_PAD_PGM_PARMS Values

Values	Description
YES	All IN and INOUT character parameter values are left justified and padded with blanks for the defined precision of the parameter, before the parameters are passed to a program or stored procedure. Trailing blanks are removed after the call to a program or stored procedure.
NO	No padding is added to character parameter values (values are NULL-terminated) when passing parameters to programs or stored procedures. Trailing blanks are not removed after calling a program or stored procedure.

Examples

Example 1: Pads parameters with blanks

```
DTW_PAD_PGM_PARMS="YES"
```

DTW_SAVE_TABLE_IN

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 11. 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

Purpose

Specifies to a database language environment to assign the total number of rows in the result set to TOTAL_ROWS. The default setting is to not assign the number; therefore, DTW_SET_TOTAL_ROWS must be set to YES to use TOTAL_ROWS in your macro.

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

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.

Values

```
DTW_SET_TOTAL_ROWS="YES"|"NO"
```

Table 12. DTW_SET_TOTAL_ROWS Values

Values	Description
YES	Assigns the value of the total number of rows to the TOTAL_ROWS variable.
NO	Net.Data does not set the TOTAL_ROWS variable and TOTAL_ROWS cannot be referenced in a macro. NO is the default.

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>Your query is limited to $(TOTAL_ROWS) rows.  The query returned too many rows.
%}
%}
```

LOGIN

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.

OS/400 ignores both LOGIN and PASSWORD if the DATABASE variable is not defined or if it is set to a value of "*LOCAL". Database access is routed through the user profile under which Net.Data is running.

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 13. 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:  <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

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 14. NULL_RPT_FIELD Values

Values	Description
null_char	Specifies a string 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

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.

OS/400 ignores both LOGIN and PASSWORD if the DATABASE variable is not defined or if it is set to a value of "*LOCAL". Database access is routed through the user profile under which Net.Data is running.

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 15. 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: <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

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. SHOWSQL can only be used if DTW_SHOWSQL is set to YES in the Net.Data configuration file. For more information about the DTW_SHOWSQL configuration variable, see the configuration chapter in *Net.Data Administration and Programming Guide* for your operating system.

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

Values

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

Table 16. 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.

Restriction: SHOWSQL generates HTML markup. When used within other presentation statements, such as JavaScript, syntax errors can occur.

Examples

Example 1: Displays all SQL queries

In the configuration file:

```
DTW_SHOWSQL YES
```

In the macro:

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

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

In the configuration file:

```
DTW_SHOWSQL YES
```

In the macro:

```
SHOWSQL: <input type="radio" name="showsql" value="yes" /> Yes
         <input type="radio" name="showsql" value="" checked /> No
```

SQL_STATE

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

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.

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

If you access multiple databases from Net.Data on OS/400 or using IBM's DataJoiner, you can achieve multiple database update coordination and consistency when updating from Net.Data.

Setting `TRANSACTION_SCOPE = "MULTIPLE"` causes all IBM database updates issued from a single HTML block to be committed or rolled back together.

Values

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

Table 17. 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

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).</p>
```

DTW_CURRENT_LAST_MODIFIED

Purpose

The date and time the current file was last modified. The current file can be a Net.Data macro 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).</p>
```

DTW_DEFAULT_MESSAGE

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.

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

Examples

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 error message with additional information, if a function returns a return code other than 0.

DTW_MACRO_FILENAME

Purpose

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

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).</p>
```

DTW_MACRO_LAST_MODIFIED

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).</p>
```

DTW_MP_PATH

Purpose

The content of this variable specifies the full path and filename of the Net.Data executable.

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

Purpose

The version and release number of Net.Data running on the server.

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

Purpose

Specifies whether or not Net.Data will automatically supply the HTTP 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.

If you are using DTW_PRINT_HEADER to generate your own headers (DTW_PRINT_HEADER = "NO"), you must either ensure that DTW_REMOVE_WS is set to "NO" or use the DTW_rHEXTOCHAR() built-in function to generate a new line after the HTTP headers.

Specify the value of this variable using a DEFINE statement, or if outside of a block you can specify it using the @DTW_ASSIGN() function.

Values

```
DTW_PRINT_HEADER="YES"|"NO"
```

Table 18. DTW_PRINT_HEADER Values

Values	Description
YES	Net.Data prints out the text `Content-type: text/HTML` or `Content-type: text/xml` 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

Example 1: Setting DTW_PRINT_HEADER to NO to customize your own header.

```
%define DTW_REMOVE_WS="YES"
%define DTW_PRINT_HEADER="NO"
@DTW_ASSIGN(CRLF, "@DTW_rHEXTOCHAR("0D25")")
%HTML(report) {Expires: Thu, 31 Jan 2001 16:00:00 GMT$(CRLF)
Content-type: text/wml$(CRLF)$(CRLF)
...
%}
```

DTW_REMOVE_WS

Purpose

If you set the value of this variable to "YES" in the DEFINE block, Net.Data will remove extra white space in the resulting web pages. Use this variable with caution when the formatting of the output requires that the whitespace not be changed, for example:

- Text between <pre></pre> tags.
- Languages other than HTML, such as the JavaScript.

Values

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

Table 19. DTW_REMOVE_WS Values

Values	Description
YES	Net.Data compresses a sequence of two or more white spaces that contain a newline character to one new-line character, and a sequence of two or more white spaces that ***does not*** contain a newline character to one space character. If the Net.Data macro contains Javascript statements which contain strings with sequences of two or more white spaces, Net.Data compresses the string constants to one white space.
NO	Net.Data does not compress white spaces. NO is the default.

Examples

Example 1: Removing extraneous white space

```
DTW_REMOVE_WS="YES"
```

RETURN_CODE

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.

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

Examples

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 103.
- **Math functions** perform mathematical operations. See "Math functions" on page 133.
- **String-manipulation functions** modify strings and characters. See "String functions" on page 153.
- **Word-manipulation functions** modify words or sets of words. See "Word functions" on page 178.
- **Table-manipulation functions** help you generate forms and reports from your table data. See "Table functions" on page 189.
- **Flat-file interface functions** perform file input and output. See "Flat file interface functions" on page 228.
- **Web-registry functions** perform operations on a Web registry. See "Web registry functions" on page 258.
- **Persistent macro functions** support transaction processing in Net.Data. See "Persistent macro functions" on page 274

Although some function parameters are described as having type *integer* or *float*, the terms 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.

Using the `DTW` prefix for functions that are not Net.Data built-in functions may result in unpredictable behavior.

Built-in function 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 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).
  ```

- Functions beginning with DTWA_ have no output parameters.

Function parameter rules

Place function parameters in the correct order. You must 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

Purpose

Replaces single quotes in an input string with two single quotes.

Format

@DTW_ADDQUOTE(stringIn, stringOut)

@DTW_rADDQUOTE(stringIn)

@DTW_mADDQUOTE(stringMult, stringMult2, ..., stringMultn)

Parameters

Table 20. 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.

Return codes

Table 21. DTW_ADDQUOTE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

1. Consider using this function for all SQL INPUT statements where input is obtained from a Web browser. 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')
```

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_DATATYPE

Purpose

Determine if a variable represents a table, numeric string, or character data.

Format

@DTW_DATATYPE(var, type)
@DTW_rDATATYPE(var)

Parameters

Table 22. DTW_DATATYPE Parameters

Data Type	Parameter	Use	Description
any type	*var*	IN	A variable or literal string.
string	*typet*	OUT	A variable that contains "NUM" if var represents an integer, the value "TABLE" if var represents a table, or the value "STRING" if var represents character data.

Return codes

Table 23. DTW_DataType Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

1. If the value of the input variable contains whitespace, the variable will not be considered an integer.
2. An integer can be preceeded by a plus (+) or minus (-) sign.

Examples

Example 1:

```
@DTW_DATATYPE(inputval, type)
%IF (type == "NUM")
 @handleNumeric(inputval)
%ELIF (type == "STRING")
 @handleString(inputval)
%ELSE
```

Error: input must be a numeric or string value

```
%ENDIF
```

DTW_DATE

Purpose

Returns the current system date in the specified format.

Format

@DTW_DATE(format, stringOut)
@DTW_DATE(stringOut)
@DTW_rDATE(format)
@DTW_rDATE()

Parameters

Table 24. 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.

Return codes

Table 25. DTW_DATE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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").</p>
```

- Returns: `04/25/97`

DTW_EXIT

Purpose

Specifies to leave the macro immediately. Net.Data sends any Web pages that are generated prior to DTW_EXIT() being called to the Web browser .

Format

@DTW_EXIT()

Return codes

Table 26. DTW_EXIT Return Codes

Return Code	Explanation
1003	An incorrect number of parameters were passed on a function call.

Usage Notes

1. Use DTW_EXIT() to immediately stop the processing of a macro. Using this technique saves the time Net.Data would use to process the entire file.
2. Ensure that the entire macro is syntactically correct before adding the DTW_EXIT() function. Using DTW_EXIT() causes Net.Data to stop processing the macro when it encounters the call to this function, which can prevent you from catching errors that occur after the DTW_EXIT() function has been processed.

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

Purpose

Returns the value of the specified cookie.

Format

@DTW_GETCOOKIE(IN cookie_name, OUT cookie_value)
@DTW_rGETCOOKIE(IN cookie_name)

Parameters

Table 27. 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. If the cookie value has URL style encodings (for example "%20"), the cookie value is decoded before the value is returned.

Return codes

Table 28. DTW_GETCOOKIE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
8000	The cookie cannot be found.

Usage Notes

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.

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

```
%MESSAGE {
    8000 : "" : continue
%}

%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" /></p>
  </form>
  %ELSE
  <p>Hi, $(name). Welcome back.</p>
  %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 form 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>
  <p><a href="welcome">Come back</a></p>
  </body>
  </HTML>
  %}
```

DTW_GETENV

Purpose

Returns the value of the specified environment variable.

Format

@DTW_GETENV(envVarName, envVarValue)
@DTW_rGETENV(envVarName)

Parameters

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*. An empty string is returned if the value is not found.

Return codes

Table 30. DTW_GETENV Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

You can also use the ENVVAR statement to reference the values of environment variables. For more information, see "ENVVAR statement" on page 13.

Examples

Example 1: Returns the value for the PATH statement on the OUT parameter

```
@DTW_GETENV(myEnvVarName, myEnvVarValue)
```

- Input: `myEnvVarName = "PATH"`
- Returns: `myEnvVarValue = "/usr/bin"`

Example 2: Returns the value for the protocol of the server

```
<p>The server is @DTW_rGETENV("SERVER_PROTOCOL").</p>
```

Returns:

The server is "HTTP/1.0".

DTW_GETINIDATA

Purpose

Returns the value of the specified configuration variable.

Format

@DTW_GETINIDATA(iniVarName, iniVarValue)

@DTW_rGETINIDATA(iniVarName)

Parameters

Table 31. 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*.

Return codes

Table 32. DTW_GETINIDATA Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

1. If a configuration variable is specified that is not the configuration file, Net.Data returns an empty string.
2. ENVIRONMENT statements cannot be retrieved with this call.

Examples

Example 1: Returns the Net.Data path variable value.

```
myEnvVarName = "FFI_PATH"
@DTW_GETINIDATA(myEnvVarName, myEnvVarValue)
```

Yields: `myEnvVarValue = "D:\FFI"`

DTW_HTMLENCODE

Purpose

Encodes selected characters using HTML character escape codes.

Format

@DTW_HTMLENCODE(stringIn, stringOut)
@DTW_rHTMLENCODE(stringIn)

Parameters

Table 33. 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 HTML character escape codes.

Return codes

Table 34. DTW_HTMLENCODE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

1. Use this function to encode character data that you do not want the Web browser to interpret as HTML. For example, by using the appropriate escape code, you can display characters such as less-than (<) and greater-than (>) within a Web page, which would otherwise be interpreted by the browser as components of HTML tags.
2. Table 35 shows the characters that are encoded by the DTW_HTMLENCODE function.

Table 35. Character Escape Codes for HTML

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;

Table 35. Character Escape Codes for HTML (continued)

:	Colon	&#58;
;	Semicolon	&#59;
<	Less than	&#60;
=	Equals	&#61:
>	Greater than	&#62:
?	Question mark	&#63:
@	At sign	&#64;
/	Backslash	&#92;
^	Carat	&#94;
{	Left brace	&#123;
|	Straight line	&#124;
}	Right brace	&#125;
~	Tilde	&#126;

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_LOG_ERRORMSG

Purpose

Allows you to write a message to the error log.

Format

@DTW_LOG_ERRORMSG(stringIn)

Parameters

Table 36. DTW_LOG_ERRORMSG Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable, literal string, or a combination.

Return codes

Table 37. DTW_LOG_ERRORMSG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.

Usage Notes

Use this function to write messages to the error log. One example would be to use a message block to catch a message that would otherwise be displayed on the screen, and to then use this function to write a customized message to the error log.

This function will only work when error logging is enabled. See the *Net.Data Administration and Programming Guide* for information on enabling the error log. The format of the message that is written to the log will have the same format as the Net.Data error messages written to the log. If tracing is also activated, this function will write the error in both the error and trace log.

Examples

Report the error code and function in which it was generated.

```
@DTW_LOG_ERRORMSG("Error occured in myfunc(), errorcode=$(myerrcode)")
```

DTW_LOG_TRACEMSG

Purpose

Allows you to write a message to the trace log.

Format

@DTW_LOG_TRACEMSG(stringIn)

Parameters

Table 38. DTW_LOG_TRACEMSG Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable, literal string, or a combination.

Return codes

Table 39. DTW_LOG_TRACEMSG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.

Usage Notes

Use this function to write messages to the trace log. One example would be to aid in debugging your application, or to provide extra information to someone servicing your application.

This function will only work when trace logging is activated. See *Net.Data Administration and Programming Guide* for information on activating the trace log. If error logging is also activated, the errors will logged in the trace log as well as the error log.

Examples

Report the current state of the variables at a given point in a function.

```
@DTW_LOG_TRACEMSG("Checkpoint 1:  Var1='$(var1)' Var2='$(var2)'")
```

DTW_QHTMLENCODE

Purpose

Performs the same function as @DTW_HTMLENCODE but also encodes the single-quote character (') as '. The HTML character escape codes that DTW_QHTMLENCODE uses are shown in Table 35 on page 114.

Format

@DTW_QHTMLENCODE(stringIn, stringOut)

@DTW_rQHTMLENCODE(stringIn)

Parameters

Table 40. 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 HTML character escape codes.

Return codes

Table 41. DTW_QHTMLENCODE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

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

Purpose

Dynamically builds and transmits electronic mail (e-mail) messages.

Format

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject, IN CarbonCopy, IN BlindCarbonCopy, IN ReplyTo, IN Organization, IN AttachList, IN AttachConvList, IN SubType)

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject, IN CarbonCopy, IN BlindCarbonCopy, IN ReplyTo, IN Organization, IN AttachList, IN AttachConvList)

@DTW_SENDMAIL(IN Sender, IN Recipient, IN Message, IN Subject, IN CarbonCopy, IN BlindCarbonCopy, IN ReplyTo, IN Organization, IN AttachList)

@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, IN Organization, IN Attachments, IN AttachConvList)

@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)

Parameters

Table 42. 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
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. This is an optional parameter. You must specify a null string ("") to specify additional parameters.

Table 42. DTW_SENDMAIL Parameters (continued)

Data Type	Parameter	Use	Description
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. This is an optional parameter. You must specify a null string ("") to specify additional parameters.
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. This is an optional parameter. You must specify a null string ("") to specify additional parameters.
string	*ReplyTo*	IN	A variable or literal string that contains the e-mail address to which replies to this message should be sent. This is an optional parameter. You must specify a null string ("") to specify additional parameters. 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*. This is an optional parameter.
string	*AttachList*	IN	List of comma-delimited fields where each field represents the file that is to be sent as an attachment. The files will be searched for in the DTW_ATTACHMENT_PATH configuration variable. A field in the list may be empty (for example, 2 consecutive commas). If the value is not set or no files are specified, then no attachments are sent.

Table 42. DTW_SENDMAIL Parameters (continued)

Data Type	Parameter	Use	Description
string	*AttachConvList*	IN	List of comma-delimited fields where each field indicates whether the data in the corresponding attachment in ***AttachList*** should be converted to ASCII or not. Possible values are: • Y - Yes, convert the data in the attachment to ASCII • N - No, do not convert the data to ASCII The default is No. A field in the list may be empty (for example, 2 consecutive commas), in which case the data in the corresponding attachment in *AttachList* will be treated as binary data (i.e. no data conversions will take place) except for the case when the file is determined to be textual, in which case the data in the file will be converted. If *AttachConvList* is not specified, the data in all attachments will be treated as binary data, except, as indicated previously, when the file is determined to be textual, in which case the data in the file will be converted. This parameter is ignored if *AttachList* is not set. See usage notes for further details.
string	*SubType*	IN	A variable or literal string that contains the content-type subtype of the message that is to be sent. If the value is not set the message content-type will be set by Net.Data to `text/plain` (the default subtype is `plain`).

Return codes

Table 43. DTW_SENDMAIL Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
7000	Net.Data is unable to connect to the specified SMTP server.
7001	An SMTP error occurred while Net.Data tried to relay the e-mail message to the specified SMTP server.
7002	The specified SMTP server does not support the Extended Simple Mail Transfer Protocol (ESMTP).

Usage Notes

1. You can use the optional configuration variable, DTW_SMTP_SERVER, to specify the SMTP server to use for transmitting e-mail messages. The value of this parameter can either be a hostname 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 this variable.

2. You can use the optional configuration variable, DTW_SMTP_CHARSET to specify which ASCII character set to use when converting the message from EBCDIC to ASCII. If DTW_SMTP_CHARSET is not specified, the default character set is iso-8859–1. See the configuration chapter in *Net.Data Administration and Programming Guide* to learn more about this variable and the supported character sets.
3. The DTW_SENDMAIL function will set the content-type of attachments based on the suffix of the attachment. If the suffix of the attachment is not recognized, then application/octet-stream will be used. Below is the mappings that Net.Data will use when mapping a file suffix to a content-type:

Extension(s)	**MIME type/subtype**	**Description/Comments**
GIF	image/gif	GIF image
BMP	image/bmp	Bitmap image
ICO	image/x-icon	Icon image
JPE JPEG JPG	image/jpeg	JPEG image
TIF TIFF	image/tiff	TIFF image
PDF	application/pdf	Acrobat
EXE	application/x-msdownload	Windows executable
PS EPS	application/postscript	Postscript
JS	application/x-javascript	Javascript
JAR	application/java-archive	Java archive
ZIP	application/zip	Zip
RTF	application/rtf	Rich text file
123 WK1 WK3 WK4	application/vnd.lotus-1-2-3	Lotus 1-2-3.
PRZ PRE	application/vnd.lotus-freelance	Lotus Freelance
LWP SAM MWP SMM	application/vnd.lotus-wordpro	Lotus WordPro
XLS	application/vnd.ms-excel	Microsoft Excel
PPT PPS	application/vnd.ms-powerpoint	Microsoft PowerPoint
DOC	application/msword	Microsoft Word
HTM HTML SHTML SHT STM HTX HTW	text/html	HTML
XML	text/xml	XML doc
XSL	text/xsl	XSL doc
TXT TEXT MBR	text/plain	Plain text. MBR is AS/400-specifiec.
CSS	text/css	Cascading style sheet
MID MIDI RMI	audio/mid	MIDI
MP3	audio/mpeg	Audio MPEG
WAV	audio/wav	Wave
MPE MPEG MPG	video/mpeg	Video MPEG
MOV	video/quicktime	Quicktime
AVI	video/avi	AVI

4. If data in an attachment is to be converted to ASCII, the OS/400 platform will convert the data in the file from the coded character set identifier (CCSID) of the file to the CCSID specified in the

configuration variable DTW_SMTP_CCSID. See the configuration chapter in the *Net.Data Administration and Programming Guide* for more information about the DTW_SMTP_CHARSET and DTW_SMTP_CCSID configuration variables.

Examples

Example 1: Function call that builds and sends a simple e-mail message

```
@DTW_SENDMAIL("<ibmuser1@ibm.com>", "<ibmuser2@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: <ibmuser2@ibm.com>
From: <ibmuser1@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("IBM User 1 <ibmuser1@ibm.com>", "IBM User 2 <ibmuser2@ibm.com>,
IBM User 3 <ibmuser3@ibm.com>, IBM User 4 <ibmuser4@ibm.com>", "There is a
meeting at 9:30.", "Status meeting", "IBM User 5 <ibmuser5@ibm.com>",
"IBM User 6 <ibmuser6@ibm.com", "meeting@ibm.com", "IBM")
```

The DTW_SENDMAIL function sends an e-mail message with the following information:

```
Date: Mon, 3 Apr 1998 09:54:33 PST
To: IBM User 2 <ibmuser2@ibm.com>, IBM User 3 <ibmuser3@ibm.com>,
    IBM User 4 <ibmuser4@ibm.com>
CC: IBM User 5 <ibmuser5@ibm.com>
BCC: IBM User 6 <ibmuser6@ibm.com>
From: IBM User 1 <ibmuser1@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>
<p>Subject:<br /><input name="subject" /></p>
<p>Message:<br /><textarea name=message rows="20" cols="40">
</textarea></p>
<p><input type="submit" value="Send E-mail"></p>
</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.</p>
</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 STATE='CA'
    %REPORT {
      Sending product information to our customers 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

Purpose

Generates JavaScript code that sets a cookie on the client system.

Format

@DTW_SETCOOKIE(IN cookie_name, IN cookie_value, IN adv_opts)

@DTW_SETCOOKIE(IN cookie_name, IN cookie_value)

Parameters

Table 44. 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.
			
			Avoid using semicolons, commas, and spaces as a part of *cookie_value*. 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"))`

string	*adv_opts*	IN	A string that contains optional attributes, separated by semicolons, that are used to define the cookie.*

Table 44. DTW_SETCOOKIE Parameters (continued)

Data Type	Parameter	Use	Description
*The optional attributes can be: **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. When the secure option 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. For additional information about all of the advanced options, see the RFC specification for HTTP state management at http://www.ietf.org/rfc/rfc2965.txt.			

Return codes

Table 45. DTW_SETCOOKIE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.

Usage Notes

1. If the client Web browser does not support Java Script, the browser does not set the cookie.
2. Because DTW_SETCOOKIE generates Java Script code, do not call DTW_SETCOOKIE inside a <script> or <noscript> HTML element.

3. To retrieve a cookie, use the DTW_GETCOOKIE() function. See "DTW_GETCOOKIE" on page 110 to learn how to define a cookie.
4. 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.

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")
```

Function calls should be on one line; the lines are split in this example for formatting purposes.

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 /></p>
  </form>
  %ELSE
  <p>Hi, $(name). Welcome back.</p>
  %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</h1>
  @DTW_SETCOOKIE("NDC_name", name,
   "expires=Wednesday, 01-Dec-2010 00:00:00;path=/")
  <p>Thank you.</p>
  <p><a href="welcome">Come back</a></p>
  </body>
  </HTML>
  %}
```

DTW_SETENV

Purpose

Assigns an environment variable with a specified value and returns the previous value.

Format

@DTW_SETENV(envVarName, envVarValue, prevValue)
@DTW_rSETENV(envVarName, envVarValue)

Parameters

Table 46. DTW_SETENV Parameters

Data Type	Parameter	Use	Description
string	*envVarName*	IN	A variable or literal string representing the environment variable.
string	*envVarValue*	IN	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.

Return codes

Table 47. DTW_SETENV Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

If no previous value for the environment variable is found, an empty string is returned.

Examples

Example 1: Returns the value for the previous path

```
@DTW_SETENV("PATH", "myPath", prevValue)
```

- Input: `envVarName = "PATH", envVarValue = "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: `envVarName = "PATH", envVarValue = "myPath"`
- Returns: `"myPreviousPath", PATH = "myPath"`

DTW_TIME

Purpose

Returns the current system time in the specified format.

Format

@DTW_TIME(stringIn, stringOut)
@DTW_TIME(stringOut)
@DTW_rTIME(stringIn)
@DTW_rTIME()

Parameters

Table 48. 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 X - Extended time (hh:mm:ss.ccc, using a 24-hour clock and where ccc is the number of milliseconds) 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.

Return codes

Table 49. DTW_TIME Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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().</p>
%}
```

- Returns: `This report was created 15:04:39, 01 May 1997.`

DTW_URLESCSEQ

Purpose

Replaces selected characters not allowed in a URL with their escape values, known as URL-encoded codes.

Format

@DTW_URLESCSEQ(stringIn, stringOut)

@DTW_rURLESCSEQ(stringIn)

Parameters

Table 50. DTW_URLESCSEQ Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string.
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.

Return codes

Table 51. DTW_URLESCSEQ Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

1. Use this function to pass any of the characters listed in Table 52 to another macro or HTML block.

Table 52. Character Escape Values for 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

Table 52. Character Escape Values for URLs (continued)

>	Greater than	%3E
?	Question mark	%3F
@	At sign	%40
[	Left bracket	%5B
/	Slash	%5C
]	Right bracket	%5D
^	Carat	%5E
{	Left brace	%7B
|	Straight line	%7C
}	Right brace	%7D
~	Tilde	%7E

Net.Data assumes that the URL contains ASCII characters only.

Examples

Example 1: Replaces the space and an ampersand characters in *string1* with their escape values and assigns the result to *string2*

```
@DTW_URLESCSEQ(string1,string2)
```

- Input: `string1 = "Guys & Dolls"`
- Returns: `string2 = "Guys%20%26%20Dolls"`

Example 2: Replaces space and ampersand characters with their escape codes.

```
@DTW_rURLESCSEQ("Guys & Dolls")
```

- Returns: `"Guys%20%26%20Dolls"`

Example 3: Uses DTW_rURLESCSEQ in a ROW block, and replaces space and 'at' characters with their escape codes.

```
%ROW{
<p><a href="fullRpt.mac/input
?name=@DTW_rURLESCSEQ(V1)&email=@DTW_rULRESCSEQ(V2)">
$(V1)</a></p>
%}
```

- 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></p>
```

When the application user clicks on the name "Patrick O'Brien," the values specified for the name and e-mail address flow within the query string of the URL that causes Net.Data to execute the input section of the fullrpt.mac macro.

Math functions

These functions let you do mathematical calculations.

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:

- 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

Purpose

Adds two numbers.

Format

@DTW_ADD(number1, number2, precision, result)
@DTW_ADD(number1, number2, result)
@DTW_rADD(number1, number2, precision)
@DTW_rADD(number1, number2)

Parameters

Table 53. 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*.

Return codes

Table 54. DTW_ADD Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.
4002	The result of an arithmetic operation had an exponent that was outside the supported range of -999,999,999 to +999,999,999.

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

Purpose

Divides one number by the other.

Format

@DTW_DIVIDE(number1, number2, precision, result)
@DTW_DIVIDE(number1, number2, result)
@DTW_rDIVIDE(number1, number2, precision)
@DTW_rDIVIDE(number1, number2)

Parameters

Table 55. DTW_DIVIDE Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number that is to be divided.
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*.

Return codes

Table 56. DTW_DIVIDE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.
4002	The result of an arithmetic operation had an exponent that was outside the supported range of -999,999,999 to +999,999,999.

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

Purpose

Divides one number by the other and returns the remainder.

Format

@DTW_DIVREM(number1, number2, precision, result)
@DTW_DIVREM(number1, number2, result)
@DTW_rDIVREM(number1, number2, precision)
@DTW_rDIVREM(number1, number2)

Parameters

Table 57. DTW_DIVREM Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number that is to be divided.
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*.

Return codes

Table 58. DTW_DIVIDEREM Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.
4002	The result of an arithmetic operation had an exponent that was outside the supported range of -999,999,999 to +999,999,999.

Usage Notes

The sign of the remainder, if nonzero, is the same as that of the first parameter.

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_EVAL

Purpose

Evaluate a mathematical expression.

Format

@DTW_EVAL(expression, result)

@DTW_rEVAL(expression)

Parameters

Table 59. DTW_EVAL Parameters

Data Type	Parameter	Use	Description
string	*expression*	IN	A variable or literal string representing a mathematical expression.
float	*result*	OUT	A variable that contains the result of evaluating the mathematical expression.

Return codes

Table 60. DTW_EVAL Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4003	The mathematical expression is not valid.
4004	An attempt was made to divide a number by zero.

Usage Notes

1. Valid operators are plus (+), minus (-), multiply (*), divide (/), power (**), and modulus (%). Any other characters, except for numbers and whitespace, are not allowed.
2. Standard precedence rules will be used. Precedence order can be changed by use of parentheses.
3. The result of mathematical operations is undefined if integers exceed allowed limits for signed 32-bit integers or floating-point numbers.
4. The operand that follows a modulus (%) or power (**) operator must be an integer.
5. The plus (+) and minus (-) operators can be unary operators, as in -3 or +10.
6. Floats can be expressed in scientific notation.

Examples

Example 1:

```
%define calcsum = "($(n) * ($(n) + 1)) / 2"

 @DTW_ASSIGN(n, "100")
 @DTW_EVAL(calcsum, sum)
```

Example 2:

```
%define forumula1 = "(-$(b) + ($(b)**2 - 4*$(a)*$(c))) / (2*$(a))"
%define forumula2 = "(-$(b) - ($(b)**2 - 4*$(a)*$(c))) / (2*$(a))"

 ...

@DTW_ASSIGN(a, "1")
@DTW_ASSIGN(b, "2")
@DTW_ASSIGN(c, "1")
@DTW_EVAL(formula1, solution1)
@DTW_EVAL(formula2, solution2)
```

DTW_FORMAT

Purpose

Customizes the formatting for a number.

Format

@DTW_FORMAT(number, before, after, expp, expt, precision, result)
@DTW_FORMAT(number, before, after, expp, expt, result)
@DTW_FORMAT(number, before, after, expp, result)
@DTW_FORMAT(number, before, after, result)
@DTW_FORMAT(number, before, result)
@DTW_FORMAT(number, result)
@DTW_rFORMAT(number, before, after, expp, expt, precision)
@DTW_rFORMAT(number, before, after, expp, expt)
@DTW_rFORMAT(number, before, after, expp)
@DTW_rFORMAT(number, before, after)
@DTW_rFORMAT(number, before)
@DTW_rFORMAT(number)

Parameters

Table 61. 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.
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.

Return codes

Table 62. DTW_FORMAT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.

Table 62. DTW_FORMAT Return Codes (continued)

Return Code	Explanation
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.

Usage Notes

1. If *number* is the only parameter is specified, the result is formatted as if @DTW_rADD(number,"0") was executed.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.

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

Purpose

Divides one number by the other and returns the integer part of the result.

Format

@DTW_INTDIV(number1, number2, precision, result)
@DTW_INTDIV(number1, number2, result)
@DTW_rINTDIV(number1, number2, precision)
@DTW_rINTDIV(number1, number2)

Parameters

Table 63. DTW_INTDIV Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number a number that is to be divided.
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*.

Return codes

Table 64. DTW_INTDIV Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.
4002	The result of an arithmetic operation had an exponent that was outside the supported range of -999,999,999 to +999,999,999.

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

Purpose

Multiplies two numbers.

Format

@DTW_MULTIPLY(number1, number2, precision, result)
@DTW_MULTIPLY(number1, number2, result)
@DTW_rMULTIPLY(number1, number2, precision)
@DTW_rMULTIPLY(number1, number2)

Parameters

Table 65. 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*.

Return codes

Table 66. DTW_MULTIPLY Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.
4002	The result of an arithmetic operation had an exponent that was outside the supported range of -999,999,999 to +999,999,999.

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

Purpose

Raises a whole number to a whole number power.

Format

@DTW_POWER(number1, number2, precision, result)
@DTW_POWER(number1, number2, result)
@DTW_rPOWER(number1, number2, precision)
@DTW_rPOWER(number1, number2)

Parameters

Table 67. DTW_POWER Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number that is to be raised to a power.
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*.

Return codes

Table 68. DTW_POWER Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.
4002	The result of an arithmetic operation had an exponent that was outside the supported range of -999,999,999 to +999,999,999.

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

Purpose

Subtracts one number from the other number.

Format

@DTW_SUBTRACT(number1, number2, precision, result)
@DTW_SUBTRACT(number1, number2, result)
@DTW_rSUBTRACT(number1, number2, precision)
@DTW_rSUBTRACT(number1, number2)

Parameters

Table 69. DTW_SUBTRACT Parameters

Data Type	Parameter	Use	Description
float	*number1*	IN	A variable or literal string representing a number from which another number is to be subtracted.
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*.

Return codes

Table 70. DTW_SUBTRACT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
4000	A parameter contains an invalid whole number value.
4001	A parameter contains an invalid number value.
4002	The result of an arithmetic operation had an exponent that was outside the supported range of -999,999,999 to +999,999,999.

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

Purpose

Assigns a value to a variable.

Format

@DTW_ASSIGN(stringOut, stringIn)

Parameters

Table 71. 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.

Return codes

Table 72. DTW_ASSIGN Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

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_CHARTOHEX

Purpose

Converts each character in a string to two hexadecimal characters.

Format

@DTW_CHARTOHEX(stringIn, stringOut)
@DTW_rCHARTOHEX(stringIn)

Parameters

Table 73. DTW_CHARTOHEX Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string that is to be converted.
string	*stringOut*	OUT	A variable that contains *stringIn* represented in hexadecimal format.

Return codes

Table 74. DTW_CHARTOHEX Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

Each hexadecimal character represents 4-bits of the input character (a character is represented by 8 bits).

Examples

Example 1: EBCDIC operating systems

```
@DTW_rCHARTOHEX("12345")
```

- Returns: `"F1F2F3F4F5"`

Example 2: ASCII operating systems

```
@DTW_rCHARTOHEX("12345")
```

- Returns: `"3132333435"`

DTW_CONCAT

Purpose

Concatenates two strings.

Format

@DTW_CONCAT(stringIn1, stringIn2, stringOut)
@DTW_rCONCAT(stringIn1, stringIn2)

Parameters

Table 75. 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*.

Return codes

Table 76. DTW_CONCAT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

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

Purpose

Deletes a substring of a 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)

Parameters

Table 77. 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*	IN	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*.

Return codes

Table 78. DTW_DELSTR Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

Examples

Example 1:

```
@DTW_DELSTR("abcde", "3", "2", result)
```

- Returns: `result = "abe"`

Example 2:

```
@DTW_rDELSTR("abcde", "4", "1")
```

- Returns: `"abce"`

DTW_HEXTOCHAR

Purpose

Converts each hexadecimal character in a string to a character value.

Format

@DTW_HEXTOCHAR(stringIn, stringOut)
@DTW_rHEXTOCHAR(stringIn)

Parameters

Table 79. DTW_HEXTOCHAR Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string that is to be converted.
string	*stringOut*	OUT	A variable that contains *stringIn* represented in character format.

Return codes

Table 80. DTW_HEXTOCHAR Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

Usage Notes

Each hexadecimal character in the input string represents 4 bits in the resultant character string (a character is represented by 8 bits). The input string must contain an even number of hexadecimal characters and can contain the following characters: 0-9, A-F, and a-f.

Examples

Example 1: EBCDIC operating systems

```
@DTW_rHEXTOCHAR("F1F2F3")
```

- Returns: `"123"`

Example 2: ASCII operating systems

```
@DTW_rHEXTOCHAR("313233")
```

- Returns: `"123"`

DTW_INSERT

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)

Parameters

Table 81. 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*.

Return codes

Table 82. DTW_INSERT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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_ISNUMERIC

Purpose

Determines if a string represents an integer.

Format

@DTW_ISNUMERIC(var, result)

@DTW_rISNUMERIC(var)

Parameters

Table 83. DTW_ISNUMERIC Parameters

Data Type	Parameter	Use	Description
string	*var*	IN	A variable or literal string.
string	*results*	IN	A variable that contains the value "YES" if *var* represents an integer, or "NO" if *var* does not represent an integer.

Return codes

Table 84. DTW_ISNUMERIC Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

1. If the value of the input variable contains whitespace, the variable will not be considered an integer.
2. An integer can be preceded by a plus (+) or minus (-) sign.

Examples

Example 1:

```
%IF (@DTW_rISNUMERIC(inputval) == "yes")
 @sqlcall(inputval)
%ELSE

Error: input must be a numeric value
%ENDIF
```

DTW_LASTPOS

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)

Parameters

Table 85. 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.

Return codes

Table 86. DTW_LASTPOS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns the length of a string.

Format

@DTW_LENGTH(stringIn, length)
@DTW_rLENGTH(stringIn)

Parameters

Table 87. 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*.

Return codes

Table 88. DTW_LENGTH Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Examples

Example 1:

```
@DTW_LENGTH("abcdefgh", result)
```

- Returns: `result = "8"`

Example 2:

```
@DTW_rLENGTH("")
```

- Returns: `"0"`

DTW_LOWERCASE

Purpose

Returns a string in all lowercase.

Format

@DTW_LOWERCASE(stringIn, stringOut)

@DTW_rLOWERCASE(stringIn)

@DTW_mLOWERCASE(stringMult1, stringMult2, ..., stringMultn)

Parameters

Table 89. 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.

Return codes

Table 90. DTW_LOWERCASE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

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_PAD

Purpose

Pads a string with a specified character.

Format

@DTW_PAD(option, var, length, padValue, result)
@DTW_PAD(option, var, length, result)
@DTW_rPAD(option, var, length, padValue)
@DTW_rPAD(option, var, length)

Parameters

Table 91. DTW_PAD Parameters

Data Type	Parameter	Use	Description
string	*option*	IN	Specifies which direction to pad the string. Possible values: • **L or l** – Pads to the left of the input string *var*. • **R or r** – Pads to the right of the input string *var*.
string	*vart*	IN	A variable or literal string that is to be padded.
integer	*length*	IN	The number of pad characters that will be used to pad the input string *var*. Value must be zero or greater.
string	*padValue*	IN	The pad character. Must be 1 byte. If not specified, the default pad character is the space (blank) character.
string	*result*	OUT	A variable that will contain *var* that is padded to the left or right with length pad characters.

Return codes

Table 92. DTW_PAD Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

Examples

Example 1:

```
@DTW_LENGTH(inputval, len)
%IF (len != "10")
   @DTW_PAD("R", inputval, @DTW_rSUBTRACT("10", len), inputval)
   @sqlcall(inputval)
%ENDIF)
```

DTW_POS

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)

Parameters

Table 93. 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.

Return codes

Table 94. DTW_POS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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_REPLACE

Purpose

Replaces characters in a string.

Format

@DTW_REPLACE(stringIn, stringFrom, stringTo, n, option, stringOut)
@DTW_REPLACE(stringIn, stringFrom, stringTo, n, stringOut)
@DTW_REPLACE(stringIn, stringFrom, stringTo, stringOut)
@DTW_rREPLACE(stringIn, stringFrom, stringTo, n, option)
@DTW_rREPLACE(stringIn, stringFrom, stringTo, n)
@DTW_rREPLACE(stringIn, stringFrom, stringTo)

Parameters

Table 95. DTW_REPLACE Parameters

Data Type	Parameter	Use	Description
string	*stringIn*	IN	A variable or literal string that is to be searched.
string	*stringFrom*	IN	A variable or literal string that is to be replaced.
string	*stringTo*	IN	A variable or literal string that replaces occurrences of *stringFrom*.
integer	*n*	IN	The position of the character at which to begin the search.
string	*option*	IN	Specifies whether to replace all occurrences, or just the first occurrence, and can have one of the following values: **A or a** Replaces all occurrences. The default is A. **F or f** Replaces only the first occurrence.
string	*stringOut*	OUT	A variable that contains *stringIn* with occurrences of *stringFrom* replaced by *stringTo*.

Return codes

Table 96. DTW_REPLACE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Examples

Example 1:

```
@DTW_rREPLACE("ABCABCABC", "AB", "1234")
```

- Returns: `"1234C1234C1234C"`

DTW_REVERSE

Purpose

Reverses a string so that the last character is first, second to last is second, until the entire string is reversed.

Format

@DTW_REVERSE(stringIn, stringOut)

@DTW_rREVERSE(stringIn)

Parameters

Table 97. 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*.

Return codes

Table 98. DTW_REVERSE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

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

Purpose

Removes leading blanks, trailing blanks, or both from a string.

Format

@DTW_STRIP(stringIn, option, stringOut)
@DTW_STRIP(stringIn, stringOut)
@DTW_rSTRIP(stringIn, option)
@DTW_rSTRIP(stringIn)

Parameters

Table 99. 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.

Return codes

Table 100. DTW_STRIP Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns a substring of a 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)

Parameters

Table 101. 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*.

Return codes

Table 102. DTW_SUBSTR Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns a string with each character translated to another character or unchanged.

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)

Parameters

Table 103. 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*.

Return codes

Table 104. DTW_TRANSLATE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

Usage Notes

1. If *tableI*, *tableO*, and the *default* character are not in the parameter list, the *stringIn* parameter is translated to uppercase.

2. 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.

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

Purpose

Returns a string in uppercase.

Format

@DTW_UPPERCASE(stringIn, stringOut)

@DTW_rUPPERCASE(stringIn)

@DTW_mUPPERCASE(stringMult1, stringMult2, ..., stringMultn)

Parameters

Table 105. 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.

Return codes

Table 106. DTW_UPPERCASE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

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

The following functions are word functions that Net.Data supports:

- "DTW_DELWORD" on page 179
- "DTW_SUBWORD" on page 181
- "DTW_WORD" on page 183
- "DTW_WORDINDEX" on page 184
- "DTW_WORDLENGTH" on page 185
- "DTW_WORDPOS" on page 186
- "DTW_WORDS" on page 188

DTW_DELWORD

Purpose

Deletes words in a string, starting 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)

Parameters

Table 107. 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*.

Return codes

Table 108. DTW_DELWORD Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns a substring of a string, beginning at word *n* s 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)

Parameters

Table 109. 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*.

Return codes

Table 110. DTW_SUBWORD Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns the *n*the word in a string.

Format

@DTW_WORD(stringIn, n, stringOut)
@DTW_rWORD(stringIn, n)

Parameters

Table 111. 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*.

Return codes

Table 112. DTW_WORD Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns the character position of the first character in the *n*th word of a string.

Format

@DTW_WORDINDEX(stringIn, n, stringOut)
@DTW_rWORDINDEX(stringIn, n)

Parameters

Table 113. 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*.

Return codes

Table 114. DTW_WORDINDEX Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns the length of the *n*th word of a string.

Format

@DTW_WORDLENGTH(stringIn, n, stringOut)
@DTW_rWORDLENGTH(stringIn, n)

Parameters

Table 115. 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*.

Return codes

Table 116. DTW_WORDLENGTH Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

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

Purpose

Returns the word number of the first occurrence of one string within another.

Format

@DTW_WORDPOS(stringIn1, stringIn2, n, stringOut)
@DTW_WORDPOS(stringIn1, stringIn2, stringOut)
@DTW_rWORDPOS(stringIn1, stringIn2, n)
@DTW_rWORDPOS(stringIn1, stringIn2)

Parameters

Table 117. 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*.

Return codes

Table 118. DTW_WORDPOS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.

Usage Notes

Multiple blanks are treated as single blanks for comparison.

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

Purpose

Returns the number of words in a string.

Format

@DTW_WORDS(stringIn, stringOut)
@DTW_rWORDS(stringIn)

Parameters

Table 119. 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*.

Return codes

Table 120. DTW_WORDS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

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 some other programming language.

DTW_TB_APPENDROW

Purpose

Adds one or more rows to the end of a Net.Data table.

Format

@DTW_TB_APPENDROW(table, rows)

Parameters

Table 121. DTW_TB_APPENDROW Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable for which rows are appended.
integer	*rows*	IN	The number of rows to append to *table*.

Return codes

Table 122. DTW_TB_APPENDROW Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.
1010	Data was written to the table until it was full, and the remainder of the data was discarded.

Usage Notes

1. The number of columns in the table must be set before calling DTW_TB_APPENDROW(). 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.
2. You can assign values to the new rows with the DTW_TB_SETV() function after rows are appended to the table, or pass the table to a language environment for processing.
3. If there is a limit on the total number of rows allowed in the table, and the number of rows to be appended can cause the limit to be exceeded, an error is returned to the caller.

Examples

Example 1: Appends ten rows to the table

```
%DEFINE myTable = %TABLE

@DTW_TB_APPENDROW(myTable, "10")
```

DTW_TB_COLS

Purpose

Returns the number of columns in a Net.Data table.

Format

@DTW_TB_COLS(table, cols)

@DTW_TB_rCOLS(table)

Parameters

Table 123. 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*.

Return codes

Table 124. DTW_TB_COLS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Examples

Example 1: Retrieves the number of columns and assigns the value to *cols*

```
%DEFINE myTable = %TABLE
%DEFINE cols = ""
...
@FillTable(myTable)
...
@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(myTable)
...
<p>My table contains @DTW_TB_rCOLS(myTable) columns.</p>
```

DTW_TB_DELETECOL

Purpose

Deletes one or more columns from a Net.data table.

Format

@DTW_TB_DELETECOL(table, after_col, cols)

Parameters

Table 125. DTW_TB_DELETECOL Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable from which columns are to be deleted.
integer	*after_col*	IN	The number of the column after which subsequent columns are to be delted. To delete the first column, specify `0`.
integer	*cols*	IN	The number of columns to delete from *table*.

Return codes

Table 126. DTW_TB_DELETECOL Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Examples

Example 1: Deletes the third and fourth columns from the table

```
%DEFINE myTable = %TABLE

@DTW_TB_DELETECOL(myTable, "3", "2")
```

Example 2: Deletes the first column from the table

```
%DEFINE myTable = %TABLE

@DTW_TB_DELETECOL(myTable, "0", "1")
```

DTW_TB_DELETEROW

Purpose

Deletes one or more rows from a Net.Data table.

Format

@DTW_TB_DELETEROW(table, start_row, rows)

Parameters

Table 127. DTW_TB_DELETEROW Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable from which rows are to be deleted.
integer	*start_row*	IN	The row number of the first row in *table* to delete.
integer	*rows*	IN	The number of rows to delete from *table*.

Return codes

Table 128. DTW_TB_DELETEROW Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Usage Notes

The number of columns in the table must be set before calling DTW_TB_DELETEROW(). 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.

Examples

Example 1: Deletes five rows starting at row 10 of a table

```
%DEFINE myTable = %TABLE

@DTW_TB_DELETEROW(myTable, "10", "5")
```

Example 2: Deletes all of the rows of a table

```
%DEFINE myTable = %TABLE

@DTW_TB_DELETEROW(myTable, "1", @DTW_TB_rROWS(myTable))
```

DTW_TB_DLIST

Purpose

Generates an HTML definition list from a Net.Data table.

Format

@DTW_TB_DLIST(table, term, def, termstyle, defstyle, link, link_u, image, image_u)
@DTW_TB_DLIST(table, term, def, termstyle, defstyle, link, link_u, image)
@DTW_TB_DLIST(table, term, def, termstyle, defstyle, link, link_u)
@DTW_TB_DLIST(table, term, def, termstyle, defstyle, link)
@DTW_TB_DLIST(table, term, def, termstyle, defstyle)
@DTW_TB_DLIST(table, term, def, termstyle)
@DTW_TB_DLIST(table, term, def)
@DTW_TB_DLIST(table, term)
@DTW_TB_DLIST(table)

Parameters

Table 129. 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.

Return codes

Table 130. DTW_TB_DLIST Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

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></dt>
<dd>
<a href="http://www.mycompany.com/link1.html"><strong>link1text</strong></a></dd>
<dt>
<img src="http://www.mycompany.com/images/image2.gif"
alt=""><b><i>image2text</i></b></dt>
<dd>
<a href="http://www.mycompany.com/link2.html"><strong>link2text</strong></a></dd>
<dt>
<img src="http://www.mycompany.com/images/image3.gif"
alt=""><b><i>image3text</i></b></dt>
<dd>
<a href="http://www.mycompany.com/link3.html"><strong>link3text</strong></a></dd>
<dt>
<img src="http://www.mycompany.com/images/image4.gif"
alt=""><b><i>image4text</i></b></dt>
<dd>
<a href="http://www.mycompany.com/link4.html"><strong>link4text</strong></a></dd>
</dl>
```

DTW_TB_DUMPH

Purpose

Prints out the contents of a Net.Data table using the HTML <pre> tag, where each row of the table is displayed on one line.

Format

@DTW_TB_DUMPH(table)

Parameters

Table 131. DTW_TB_DUMPH Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A symbol specifying the macro table variable to display.

Return codes

Table 132. DTW_DB_DUMPH Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1007	A parameter contains a value which is not valid.

Usage Notes

If the Net.Data table is empty, an error is returned.

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

Purpose

Prints out the contents of the Net.Data table using the HTML <pre> tag, where each field of the table is on one line.

Format

@DTW_TB_DUMPV(table)

Parameters

Table 133. DTW_TB_DUMPV Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	A symbol specifying the macro table variable to display.

Return codes

Table 134. DTW_TB_DUMPV Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1007	A parameter contains a value which is not valid.

Usage Notes

If the Net.Data table is empty, an error is returned

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

Purpose

Returns a column heading from a Net.Data table.

Format

@DTW_TB_GETN(table, col, name)

@DTW_TB_rGETN(table, col)

Parameters

Table 135. DTW_TB_GETN Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable from which a column name is returned.
integer	*col*	IN	The column number of the column whose name is to be returned.
string	*name*	OUT	A variable that contains the name of the column specified in *col*.

Return codes

Table 136. DTW_TB_GETN Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Usage Notes

Before calling DTW_TB_GETN(), set the number of columns in the table. 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.

Examples

Example 1: Retrieves the column name of column 4

```
%DEFINE myTable = %TABLE
%DEFINE name = ""
...
@FillTable(myTable)
...
@DTW_TB_GETN(myTable, "4", name)
```

Example 2: Retrieves the column name of the last column in the table

```
%DEFINE myTable = %TABLE
...
@FillTable(myTable)
...
<p>The column name of the last column is @DTW_TB_rGETN(myTable,
@DTW_TB_rCOLS(myTable))</p>
```

DTW_TB_GETV

Purpose

Returns the value at a given row and column in a Net.Data table.

Format

@DTW_TB_GETV(table, row, col, value)

@DTW_TB_rGETV(table, row, col)

Parameters

Table 137. DTW_TB_GETV Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable for which a table value is returned.
integer	*row*	IN	The row number of the value to be returned.
integer	*col*	IN	The column number of the value to be returned.
string	*value*	OUT	A variable that contains the value at the row and column specified in *row* and *col*.

Return codes

Table 138. DTW_TB_GETV Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Usage Notes

Before calling DTW_TB_GETV(), set the number of columns in the table. 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.

Examples

Example 1: Retrieves the table value at row 6, column 3

```
%DEFINE myTable = %TABLE
%DEFINE value = ""
...
@FillTable(myTable)
...
@DTW_TB_GETV(myTable, "6", "3", value)
```

Example 2: Retrieves the table value at row 1, column 1

```
%DEFINE myTable = %TABLE
...
@FillTable(myTable)
...
<p>The table value of row 1, column 1 is @DTW_TB_rGETV(myTable, "1", "1").</p>
```

DTW_TB_HTMLENCODE

Purpose

Replaces certain characters in the data located in a Net.Data table with their corresponding HTML character escape codes.

Format

@DTW_TB_HTMLENCODE(table, collist)

@DTW_TB_HTMLENCODE(table)

Parameters

Table 139. 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.

Return codes

Table 140. DTW_TB_HTMLENCODE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Usage Notes

The characters that are replaced are indicated in the table below.

Name	Character	Code
Ampersand	&	&#38;
Double quote	"	&#34;
Greater than	>	&#62;
Less than	<	&#60;

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

Purpose

Generates one or more HTML check box input tags from a Net.Data table.

Format

@DTW_TB_INPUT_CHECKBOX(table, prompt, namecol, valuecol, rows, checkedrows)
@DTW_TB_INPUT_CHECKBOX(table, prompt, namecol, valuecol, rows)
@DTW_TB_INPUT_CHECKBOX(table, prompt, namecol, valuecol)
@DTW_TB_INPUT_CHECKBOX(table, prompt, namecol)

Parameters

Table 141. 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.
integer	*valuecol*	IN	The column number in *table* that contains 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.

Return codes

Table 142. DTW_TB_INPUT_CHECKBOX Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

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

Purpose

Generates HTML radio button input tags from a Net.Data table.

Format

@DTW_TB_INPUT_RADIO(table, prompt, namecol, valuecol, rows, checkedrows)
@DTW_TB_INPUT_RADIO(table, prompt, namecol, valuecol, rows)
@DTW_TB_INPUT_RADIO(table, prompt, namecol, valuecol)

Parameters

Table 143. 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.
integer	*valuecol*	IN	The column number in *table* that contains 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.

Return codes

Table 144. DTW_TB_INPUT_RADIO Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

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

Purpose

Generates HTML <input /> tags for specified rows in a Net.Data table.

Format

@DTW_TB_INPUT_TEXT(table, prompt, namecol, valuecol, size, maxlen, rows)
@DTW_TB_INPUT_TEXT(table, prompt, namecol, valuecol, size, maxlen)
@DTW_TB_INPUT_TEXT(table, prompt, namecol, valuecol, size)
@DTW_TB_INPUT_TEXT(table, prompt, namecol, valuecol)
@DTW_TB_INPUT_TEXT(table, prompt, namecol)

Parameters

Table 145. 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* that contains the input field names.
integer	*valuecol*	IN	The column number in *table* that contains 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.

Return codes

Table 146. DTW_TB_INPUT_TEXT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.

Table 146. DTW_TB_INPUT_TEXT Return Codes (continued)

Return Code	Explanation
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

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>
<p>image2text
<input type="text" name="image2text" value="link2text" size="35" maxlength="40" /></p>
<p>image3text
<input type="text" name="image3text" value="link3text" size="35" maxlength="40" /></p>
```

DTW_TB_INSERTCOL

Purpose

Inserts one or more columns into a Net.Data table.

Format

@DTW_TB_INSERTCOL(table, after_col, cols)

Parameters

Table 147. DTW_TB_INSERTCOL Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable into which columns are to be inserted.
integer	*after_col*	IN	The column number of the column after which the new columns are to be inserted. To insert columns at the beginning of the table, specify `0`.
integer	*cols*	IN	The number of columns to insert into *table*.

Return codes

Table 148. DTW_TB_INSERTCOL Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Examples

Example 1: Inserts five columns at the end of a table

```
%DEFINE myTable = %TABLE

@DTW_TB_INSERTCOL(myTable, @DTW_TB_rCOLS(myTable), "5")
```

Example 2: Inserts a column at the beginning of a table

```
%DEFINE myTable = %TABLE

@DTW_TB_INSERTCOL(myTable, "0", "1")
```

DTW_TB_INSERTROW

Purpose

Inserts one or more rows into a Net.Data table.

Format

@DTW_TB_INSERTROW(table, after_row, rows)

Parameters

Table 149. DTW_TB_INSERTROW Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable into which rows are to be inserted.
integer	*after_row*	IN	The number of the row after which new rows are inserted. To insert rows at the beginning of the table, specify `0`.
integer	*rows*	IN	The number of rows to insert into *table*.

Return codes

Table 150. DTW_TB_INSERTROW Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Usage Notes

Before calling DTW_TB_INSERTROW(), set the number of columns in the table. 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.

Examples

Example 1: Inserts a row after row five of a table

```
%DEFINE myTable = %TABLE

@DTW_TB_INSERTROW(myTable, "5", "1")
```

Example 2: Inserts three rows at the start of a table

```
%DEFINE myTable = %TABLE

@DTW_TB_INSERTROW(myTable, "0", "3")
```

DTW_TB_LIST

Purpose

Generates an HTML list from a Net.Data table.

Format

@DTW_TB_LIST(table, listtype, listitem, itemstyle, link_u, image_u)
@DTW_TB_LIST(table, listtype, listitem, itemstyle, link_u)
@DTW_TB_LIST(table, listtype, listitem, itemstyle)
@DTW_TB_LIST(table, listtype, listitem)
@DTW_TB_LIST(table, listtype)

Parameters

Table 151. 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 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.

Return codes

Table 152. DTW_TB_LIST Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.

Table 152. DTW_TB_LIST Return Codes (continued)

Return Code	Explanation
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

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>
<li><a href="http://www.mycompany.com/link2.html">
<img src="http://www.mycompany.com/images/image2.gif"
alt="">link2text</a></li>
<li><a href="http://www.mycompany.com/link3.html">
<
IMG SRC="http://www.mycompany.com/images/image3.gif"
ALT="">link3text</a></li>
<li><a href="http://www.mycompany.com/link4.html">
<img src="http://www.mycompany.com/images/image4.gif"
alt="">link4txt</a></li>
</ol>
</u></tt>
```

DTW_TB_QUERYCOLNONJ

Purpose

Returns the column number associated with a column heading of a Net.Data table.

Format

@DTW_TB_QUERYCOLNONJ(table, name, col)
@DTW_TB_rQUERYCOLNONJ(table, name)

Parameters

Table 153. DTW_TB_QUERYCOLNONJ Parameters

Data Type	Parameter	Use	Description
table	*table*	IN	The macro table variable from which a column number is to be returned.
string	*name*	IN	The name of the column heading for which the column number is returned. If the column heading does not exist in the table, `0` is returned.
integer	*col*	OUT	A variable that contains the column number of the column whose name is specified in *name*.

Return codes

Table 154. DTW_TB_QUERYCOLNONJ Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

1. Before calling DTW_TB_QUERYCOLNONJ(), set the number of columns in the table. 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.
2. If the column heading does not exist in the table, 0 is returned.

Examples

Example 1: Retrieves the column number for the column whose name is `SERIAL_NUMBER`

```
%DEFINE myTable = %TABLE
%DEFINE col = ""

@DTW_TB_QUERYCOLNONJ(myTable, "SERIAL_NUMBER", col)
```

Example 2: Retrves the column number for the column whose name is `SERIAL_NUMBER`

```
%DEFINE myTable = %TABLE
<p>The "SERIAL_NUMBER" column is column number @DTW_TB_rQUERYCOLNONJ
(myTable, "SERIAL_NUMBER")</p>
```

DTW_TB_ROWS

Purpose

Returns the number of rows in a Net.Data table.

Format

@DTW_TB_ROWS(table, rows)

@DTW_TB_rROWS(table)

Parameters

Table 155. 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*.

Return codes

Table 156. DTW_TB_ROWS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Examples

Example 1: Retrieves the current number of rows in the table and assigns the value to *rows*

```
%DEFINE myTable = %TABLE
%DEFINE rows = ""
...
@FillTable(myTable)
...
@DTW_TB_ROWS(myTable, rows)
```

DTW_TB_SELECT

Purpose

Generates an HTML selection list from a Net.Data table.

Format

@DTW_TB_SELECT(table, name, optioncol, size, multiple, rows, selectedrows, valuecol)
@DTW_TB_SELECT(table, name, optioncol, size, multiple, rows, selectedrows)
@DTW_TB_SELECT(table, name, optioncol, size, multiple, rows)
@DTW_TB_SELECT(table, name, optioncol, size, multiple)
@DTW_TB_SELECT(table, name, optioncol, size)
@DTW_TB_SELECT(table, name, optioncol)
@DTW_TB_SELECT(table, name)

Parameters

Table 157. 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	*rows*	IN	The row numbers from *table* to use in the SELECT field. The default is to use all the rows.
string	*selectedrows*	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.
string	valuecol	IN	The column number in *table* to use for the VALUE attribute of the OPTION tags. The default value is 1. This parameter is optional.

Return codes

Table 158. DTW_TB_SELECT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.

Table 158. DTW_TB_SELECT Return Codes (continued)

Return Code	Explanation
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Examples

Example 1: Generates an HTML SELECT menu with multiple selections

```
@DTW_TB_SELECT(Mytable,"URL6","4","","y","1 2 4","1 4")
```

Results:

```
<select name="url6" size="3" multiple>
<option selected>image1text
<option>image2text
<option selected>image4text
</select>
```

Example 2: Uses the *valuecol* parameter to generate an HTML SELECT menu that uses a column number from which to obtain the values.

```
@DTW_TB_SELECT(Mytable,"URL6","4","","y","1 2 4","1 4", "2")
```

Results:

```
<select name="url6" size="3" multiple>
<option value="text_string1" selected>image1text</option>
<option value="text_string2">image2text</option>
<option value="text_string4" selected>image4text</option>
</select>
```

DTW_TB_SETCOLS

Purpose

Sets the number of columns in a Net.Data table.

Format

@DTW_TB_SETCOLS(table, cols)

Parameters

Table 159. DTW_TB_SETCOLS Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable for which the number of columns is set.
integer	*cols*	IN	The initial number of columns to allocate in *table*.

Return codes

Table 160. DTW_TB_SETCOLS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.

Usage Notes

1. The DTW_TB_SETCOLS() function can only be used once for a table. Afterwards, use the DTW_TB_DELETECOL() or DTW_TB_INSERTCOL() functions to change the number of columns in the table.
2. Specify the column headings by using the DTW_TB_SETN() function.

Examples

Example 1: Allocates three columns for the table and assigns the names to the columns

```
%DEFINE myTable = %TABLE

@DTW_TB_SETCOLS(myTable, "3")
@DTW_TB_SETN(myTable, "Name", "1")
@DTW_TB_SETN(myTable, "Address", "2")
@DTW_TB_SETN(myTable, "Phone", "3")
```

DTW_TB_SETN

Purpose

Assigns a name to a column heading in a Net.Data.

Format

@DTW_TB_SETN(table, name, col)

Parameters

Table 161. DTW_TB_SETN Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable in which a column name will be set.
string	*name*	IN	A character string that is assigned to the column heading of the column specified in *col*.
integer	*col*	IN	The column number of the column whose heading is being set.

Return codes

Table 162. DTW_TB_SETN Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Usage Notes

1. Before calling DTW_TB_SETN(), set the number of columns in the table. 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.
2. To delete a column heading, assign the column heading value to NULL.

Examples

Example 1: Assigns a name to column headings 1 through 3

```
%DEFINE myTable = %TABLE

@DTW_TB_SETCOLS(myTable, "3")
@DTW_TB_SETN(myTable, "Name", "1")
@DTW_TB_SETN(myTable, "Address", "2")
@DTW_TB_SETN(myTable, "Phone", "3")
```

Example 2: Delete the column heading for column 2. This is done by passing a variable on the function call which has not been defined. By default, this variable will have a value of NULL

```
%DEFINE myTable = %TABLE

@DTW_TB_SETN(myTable, nullVar, "2")
```

DTW_TB_SETV

Purpose

Assigns a value to a particular row and column in a Net.Data table.

Format

@DTW_TB_SETV(table, value, row, col)

Parameters

Table 163. DTW_TB_SETV Parameters

Data Type	Parameter	Use	Description
table	*table*	INOUT	The macro table variable in which a table value will be set.
string	*value*	IN	A character string that is assigned to the table value of the row and column specified in *row* and *col*.
integer	*row*	IN	The row number of the value to be set.
integer	*col*	IN	The column number of the value to be set.

Return codes

Table 164. DTW_TB_SETV Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

Usage Notes

1. Before calling DTW_TB_SETV(), set the number of columns in the table. 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.
2. To delete a table value, assign the value to NULL.

Examples

Example 1: Assigns a value to row 3 column 3

```
%DEFINE myTable = %TABLE

@DTW_TB_SETV(myTable, "value3.3", "3", "3")
```

Example 2: Delete the table value at row 4, column 2. This is done by passing a variable on the function call which has not been defined. By default, this variable will have a value of NULL.

```
%DEFINE myTable = %TABLE

@DTW_TB_SETV(myTable, nullVar, "4", "2")
```

DTW_TB_TABLE

Purpose

Generates an HTML table from a Net.Data table.

Format

@DTW_TB_TABLE(table, options, collist, cellstyle, link_u, image_u, url_text, url_style)
@DTW_TB_TABLE(table, options, collist, cellstyle, link_u, image_u, url_text)
@DTW_TB_TABLE(table, options, collist, cellstyle, link_u, image_u)
@DTW_TB_TABLE(table, options, collist, cellstyle, link_u)
@DTW_TB_TABLE(table, options, collist, cellstyle)
@DTW_TB_TABLE(table, options, collist)
@DTW_TB_TABLE(table, options)
@DTW_TB_TABLE(table)

Parameters

Table 165. 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.

Return codes

Table 166. DTW_TB_TABLE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.

Table 166. DTW_TB_TABLE Return Codes (continued)

Return Code	Explanation
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

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>
<th>LINKURL</th>
<th>IMAGEURL</th>
<tr>
<td><i>link1text</i></td>
<td><a href="http://www.mycompany.com/link1.html"><b>link1text
</b></a></td>
<td><img src="http://www.mycompany.com/images/image1.gif" alt=""><b>link1text
</b></td>
</tr><tr>
<td><i>link2text</i></td>
<td><a href="http://www.mycompany.com/link2.html"><b>link2text
</b></a></td>
<td><img src="http://www.mycompany.com/images/image2.gif" alt=""><b>link2text
</b></td>
</tr><tr>
<td><i>link3text</i></td>
<td><a href="http://www.mycompany.com/link3.html"><b>link3text
</b></a></td>
<td><img src="http://www.mycompany.com/images/image3.gif" alt=""><b>link3text
</b></td>
</tr></table>
```

DTW_TB_TEXTAREA

Purpose

Generates an HTML text area from a Net.Data table.

Format

@DTW_TB_TEXTAREA(table, name, numrows, numcols, valuecol, rows)
@DTW_TB_TEXTAREA(table, name, numrows, numcols, valuecol)
@DTW_TB_TEXTAREA(table, name, numrows, numcols)
@DTW_TB_TEXTAREA(table, name, numrows)
@DTW_TB_TEXTAREA(table, name)

Parameters

Table 167. 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.

Return codes

Table 168. DTW_TB_TEXTAREA Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
1008	A parameter is outside of table bounds.

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. The following flat file interface built-in functions are available:

The following sections discuss how to use the FFI built-in functions and access flat file sources:

Access to flat file data sources

You use the FFI_PATH path configuration statement in the Net.Data initialization file to list the directories and sub-directories that are allowed to be specified when using the FFI functions and to provide security for those files not in directories included in the path statement. The Net.Data initialization file is shipped without FFI_PATH. See *Net.Data Administration and Programming Guide* to learn how to configure the path.

The FFI_PATH uses the following syntax:

```
FFI_PATH /path1;/path2;/path3...
```

When you call the FFI language environment in a macro function, you specify the path to the flat file that the FFI function is working with, using the *filename* parameter of the FFI function. For example:

```
%DEFINE myfile = "/macros/myfile.txt" @DTWF_READ(myfile, ...)
```

The following sections discuss:

How Net.Data resolves the location of flat files

Net.Data uses the information in the *filename* parameter for FFI functions to search the FFI_PATH statement in the Net.Data initialization file and determine whether to use a specified directory or the current directory.

When a file name is specified on an FFI function, Net.Data attempts to locate the file by searching each of the paths listed in FFI_PATH, starting from the first path that is specified. Net.Data uses the first copy

that it finds. If the file is not found, then Net.Data attempts to find the file in the current working directory of the process or thread in which Net.Data is running.

Example: Net.Data uses the FFI_PATH configuration statement to locate a file

The FFI_PATH contains the following directories:

```
FFI_PATH /macros;/macros/org1;/macros/org2
```

And, the file is located in both the current directory and /macros/org1. If the function call is:

```
DTWF_READ("myfile.txt")
```

Net.Data will use /macros/org1/myfile.txt.

If the DTWF_READ function is being used to read an existing file, and a file name of `myfile.txt` is specified, then Net.Data searches the directories `/macros`, `/macros/org1` and `/macros/org2` for the file, assuming that the FFI_PATH contains the list of paths specified above.

Determining the Current Directory:

The current directory for Net.Data depends on the configuration of your Web server:

- If you are using CGI, the current directory is the directory that Net.Data is running from.
- If you are using a Web server API, the current directory can vary. If the server's default request routing or resource mapping is changed, the current directory might be changed, also.

Recommendations for specifying flat file access:

Use the following recommendations to ensure that Net.Data can access flat file data sources.

- When using the DTWF_OPEN function to create flat files, ensure that you specify a directory path that is in FFI_PATH or that you know what the current directory is. If you do not specify a directory, Net.Data attempts to create the file in the current working directory.
- If you include directories in the *filename* parameter, specify the full path that matches one of the paths in FFI_PATH because Net.Data does not search sub-directories within directories specified in FFI_PATH.
- Use absolute paths for the *filename* parameter, especially if you are using a Web server API.

Flat file configuration rules

Use the following rules when adding or updating the FFI_PATH in the Net.Data initialization file:

- Path statements in FFI_PATH must contain valid printable characters. FFI does not allow paths that include a question mark (?) or double quotes (“”).
- All directories and sub-directories that are used with the *filename* parameter in the macro must be specified in the FFI_PATH. Sub-directories of the paths listed in *filename* are not searched unless explicitly specified in FFI_PATH.
- Use absolute paths for the FFI_PATH statement.

Security recommendations

You can specify which files FFI functions can access with the FFI_PATH statement in the Net.Data initialization file. FFI only searches the paths listed in the statement, so files in other directories are protected from unauthorized access.

For example, you can specify an FFI_PATH similar to the one below, designating directories for public or guest user IDs.

```
FFI_PATH     /public;/WWW;/guest;/:
```

The following list provides recommendations for making your flat files secure:

- Choose which directories are appropriate to use for flat file operations. These directories need to be added to the FFI_PATH to limit searching to those directories.
- Use care letting people perform DTWF_REMOVE or other export operations in the macro to prevent people from removing or altering files with extensions `.dll` and `.cmd` that you might have in the current directory.
- Take appropriate steps to safeguard the files on the system by using reasonable control over what macros are added to the system.
- Do not specify a path in FFI_PATH that lets anonymous FTP users write to the path. If you do, somebody can put a Net.Data macro on the system that allows actions that were not previously allowed.
- Do not add the path of the Net.Data initialization file to the FFI_PATH.

Authorization requirement

Ensure that the user ID under which Net.Data executes has access rights to files used by the FFI built-in functions. 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.

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. A *delimiter* is a flag or separator that FFI uses when dividing the file into parts (such as columns of a row) according to the requested transform. 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 line in the corresponding flat file onto a single row in the table. In this case, the new-line character 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 writes row data to a line in a flat file, it places the delimiter string as a separator between the column entries. When Net.Data rebuilds a table from a flat file, it uses the delimiter string to determine how much of each line to place in a column of the table. In this case, the regular new-line character separates the lines in the flat file that correspond to rows in the table, and the delimiter string separates the items within a single line.

For read operations, the delimiter separates the contents of the file into rows and columns of a table. For write operations, the delimiter indicates the end of a value in a table row and column. Net.Data passes the delimiter to the FFI as a Net.Data macro string and does not include a null character at the end of the characters unless explicitly listed in the DELIMITER parameter.

To use the null character in the delimiter, specify the DELIMITER parameter as a slash and a zero in double quotes, "\0", instead of an empty string by using two double quotes, """". If you specify the ASCIITEXT transform, Net.Data uses the new-line character as the delimiter and ignores any requested delimiter.

Undesirable changes to a file can occur if you use a different delimiter for write operations than for read operations. Net.Data writes the file with the new delimiter.

The maximum length of a delimiter is 256 characters.

Locking files

You can lock flat files using the DTWF_OPEN and DTWF_CLOSE functions. With these functions, Net.Data reserves a flat file so that no other applications can read or update the file.

To lock a file, use the DTWF_OPEN function. This function ensures the file is unavailable to other applications and prevents the file from changing between the time it is read and updated.

To free the file, use the DTWF_CLOSE function. This function releases the file so that other applications can read or update the file.

DTWF_APPEND

Purpose

Writes the contents of a Net.Data table to the end of a text file.

Format

@DTWF_APPEND(filename, transform, delimiter, table, retry, rows)
@DTWF_APPEND(filename, transform, delimiter, table, retry)
@DTWF_APPEND(filename, transform, delimiter, table)

Parameters

Table 169. 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. A new-line character in a file indicates the end of a row of a Net.Data macro table for ASCIITEXT and DELIMITED transforms.
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.

Return codes

Table 170. DTWF_APPEND Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.

Table 170. DTWF_APPEND Return Codes (continued)

Return Code	Explanation
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2003	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Usage Notes

The current contents of a file affect the results of using DTWF_APPEND, especially the contents of the last column of the last row. If a new-line character follows the last column value of the last row of the file, appended data is placed in a new row. Otherwise, appended data becomes part of the last row of the file. If the file to be appended does not exist, a file is created.

Examples

Example 1:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
%}
@DTWF_APPEND(myFile, "DELIMITED", " ;", myTable)
```

Example 2:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
%}
@DTWF_APPEND(myFile, "ASCIITEXT", " ;", myTable)
```

Example 3:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
%}
@DTWF_APPEND(myFile, "ASCIITEXT", " ;", myTable, "0", "10")
```

DTWF_CLOSE

Purpose

Closes a file opened by DTWF_OPEN.

Format

@DTWF_CLOSE(filename, retry)
@DTWF_CLOSE(filename)

Parameters

Table 171. 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.

Return codes

Table 172. DTWF_CLOSE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
2002	A flat file interface built-in function could not close the specified file because it was not opened by this macro invocation.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.

Examples

Example 1:

```
@DTWF_CLOSE(myFile, "5")
```

DTWF_COPY

Purpose

Copies a file.

Format

@DTWF_COPY(fromFilename, toFilename)

Parameters

Table 173. DTWF_COPY Parameters

Data Type	Parameter	Use	Description
string	*fromFilename*	INOUT	The name of the file to copy.
string	*toFilename*	INOUT	The name of the file that will contain the data in the file specified by fromFilename. If the file exists, all data in the file will be removed prior to the copy operation.

Return codes

Table 174. DTWF_COPY Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2001	A flat file interface built-in function could not open the specified file because it was in use by this or another process, and could not be shared in the specified mode.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Examples

Example 1:

```
%DEFINE {
fromFile = "/private/fromfile"
toFile = "/private/tofile"
%}
@DTWF_COPY(fromFile, toFile)
```

DTWF_DELETE

Purpose

Deletes lines from a text file.

Format

@DTWF_DELETE(filename, transform, delimiter, retry, rows, startline)
@DTWF_DELETE(filename, transform, delimiter, retry, rows)
@DTWF_DELETE(filename, transform, delimiter, retry)
@DTWF_DELETE(filename, transform, delimiter)

Parameters

Table 175. 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. A new-line character in a file indicates the end of a row of a Net.Data macro table for ASCIITEXT and DELIMITED transforms.
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	*startline*	INOUT	The line number from which to begin deleting. A value of 1 means to begin deleting at the first line. If this value is greater than the number of lines in the file, an error is returned and the value of this parameter is changed to the number of lines in the file. The default is 1.

Return codes

Table 176. DTWF_DELETE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.

Table 176. DTWF_DELETE Return Codes (continued)

Return Code	Explanation
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2003	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Examples

Example 1:

```
%DEFINE {
   myFile = "/private/myfile"
   myTable = %TABLE
   myWait = "5000"
   myRows = "2"
%}
@DTWF_DELETE(myFile, "Delimited", "|", myWait, myRows)
```

Example 2:

```
%DEFINE {
   myFile = "/private/myfile"
   myTable = %TABLE
   myStart = "1"
   myRows = "2"
%}
@DTWF_DELETE(myFile, "Asciitext", "|", "0", myRows, myStart)
```

DTWF_EXISTS

Purpose

Determines the existence of a file.

Format

@DTWF_EXISTS(filename, existenceFlag)

@DTWF_rEXISTS(filename)

Parameters

Table 177. DTWF_EXISTS Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file searched for.
string	*existenceFlag*	OUT	A variable set to show one of the following: • Y - Yes, file exists • N- No, the file does not exist

Return codes

Table 178. DTWF_EXISTS Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.

Examples

Example 1:

```
%DEFINE {
myFile = "/private/myfile"
myExistFlag=""
%}
@DTWF_EXISTS(myFile,myExistFlag)
```

Example 2:

```
%DEFINE {
myFile = "/private/myfile"
%}
@DTWF_rEXISTS(myFile)
```

DTWF_INSERT

Purpose

Inserts lines into an existing text file.

Format

@DTWF_INSERT(filename, transform, delimiter, table, retry, rows, startline)
@DTWF_INSERT(filename, transform, delimiter, table, retry, rows)
@DTWF_INSERT(filename, transform, delimiter, table, retry)
@DTWF_INSERT(filename, transform, delimiter, table)

Parameters

Table 179. 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. A new-line character in a file indicates the end of a row of a Net.Data macro table for ASCIITEXT and DELIMITED transforms.
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 lines 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	*startline*	INOUT	The line number from which to begin inserting. If this value is greater than the number of lines in the file, an error is returned and the value of this parameter is changed to the number of lines in the file. Specifying 0 means to insert starting at the beginning of the file. The default is 0.

Return codes

Table 180. DTWF_INSERT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.

Table 180. DTWF_INSERT Return Codes (continued)

Return Code	Explanation
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2003	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Examples

Example 1:

```
%DEFINE {
   myFile = "/private/myfile"
   myTable = %TABLE
   myWait = "3000"
%}
@DTWF_INSERT(myFile, "Delimited", "|", myTable, myWait)
```

Example 2:

```
%DEFINE {
   myFile = "/private/myfile"
   myTable = %TABLE
   myStart = "1"
   myRows = "2"
%}
@DTWF_INSERT(myFile, "Asciitext", "|", myTable, "0", myRows, myStart)
```

DTWF_OPEN

Purpose

Opens a text file.

Format

@DTWF_OPEN(filename, mode, retry, createOptions)

@DTWF_OPEN(filename, mode, retry)

@DTWF_OPEN(filename, mode)

Parameters

Table 181. 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.
string	*createOptions*	IN	Options to use when creating a file. If a file exists, the options specified are not used. The following option is supported: CCSID=nnn, which specifies the coded character set ID (CCSID) to use when creating a new file. Nnn must be a valid CCSID from 1 to 65534.

Return codes

Table 182. DTWF_OPEN Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Table 182. DTWF_OPEN Return Codes (continued)

Return Code	Explanation
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2001	A flat file interface built-in function could not open the specified file because it was in use by this or another process, and could not be shared in the specified mode.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Usage Notes

1. When the file does not exist, an absolute path for the filename should be specified, and the directory where the file is to be created must match a directory specified in FFI_PATH. If an absolute path is not used, the file will be opened in the current working directory.
2. DTWF_OPEN keeps the file open, otherwise, the file is closed after each flat file operation.
3. Use DTWF_OPEN to reduce the number of times a file is open. If DTWF_OPEN is not used, the file is closed after each flat file operation. The file is left open until it is closed using DTWF_CLOSE or macro processing ends.

Examples

Example:

```
%DEFINE {
  myFile = "/private/myfile"
  myMode = "r+"
%}
@DTWF_OPEN(myFile, myMode, "1000")
```

DTWF_READ

Purpose

Reads lines from a text file into a Net.Data table.

Format

@DTWF_READ(filename, transform, delimiter, table, retry, lines, startline, columns)
@DTWF_READ(filename, transform, delimiter, table, retry, lines, startline)
@DTWF_READ(filename, transform, delimiter, table, retry, lines)
@DTWF_READ(filename, transform, delimiter, table, retry)
@DTWF_READ(filename, transform, delimiter, table)

Parameters

Table 183. 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. A new-line character in a file indicates the end of a row of a Net.Data macro table for ASCIITEXT and DELIMITED transforms.
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	*lines*	INOUT	The number of lines in the file to read into the table. A value of 0 means to read to the end of the file or until the table is full; this is the default. Upon successful completion of this funciton call, the number of rows in the resulting table is returned.
integer	*startline*	IN	The line in the file from which to start reading. The default is to start reading at the first line.
integer	*columns*	OUT	Returns the number of columns in the table.

Return codes

Table 184. DTWF_READ Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.

Table 184. DTWF_READ Return Codes (continued)

Return Code	Explanation
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
1010	Data was written to the table until it was full, and the remainder of the data was discarded.
2000	A flat file interface built-in function could not find the specified file.
2003	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Examples

Example 1:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
  myWait = "1000"
%}
@DTWF_READ(myFile, "DELIMITED", ";", myTable, myWait)
```

Example 2:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
  myWait = "0"
  myRows = "0"
  myStartrow = "1"
  myColumns = ""
%}
@DTWF_READ(myFile, "DELIMITED", ";", myTable, myWait, myRows,
             myStartrow, myColumns)
```

Example 3:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
%}
@DTWF_READ(myFile, "ASCIITEXT", ";", myTable)
@DTW_TB_TABLE(myTable,"BORDER","")
```

DTWF_READFILE

Purpose

Reads a file into a Net.Data variable.

Format

@DTWF_READFILE(filename,fileData)

Parameters

Table 185. DTWF_READFILE Parameters

Data Type	Parameter	Use	Description
string	*filename*	IN	The name of the file to read into the variable. On successful completion of the function call, the fully qualified file name is returned in this variable.
string	*fileData*	OUT	The variable that is assigned the contents of the file.

Return codes

Table 186. DTWF_READFILE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2001	A flat file interface built-in function could not open the specified file because it was in use by this or another process, and could not be shared in the specified mode.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Examples

```
%define filename="/bios/$(name).txt"
%html(input) {
   <form method="get" action="report">
   <p>Name: <input name="name" /><br />
            <input type="submit" /></p>
```

```
   </form>
%}

%html(report) {
@DTWF_READFILE(filename, contents)
<p><b>Bio:</b><br />
$(contents)</p>
%}
```

DTWF_REMOVE

Purpose

Deletes an entire file.

Format

@DTWF_REMOVE(filename, retry)
@DTWF_REMOVE(filename)

Parameters

Table 187. DTWF_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.

Return codes

Table 188. DTWF_REMOVE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Examples

Example 1:

```
%DEFINE myFile = "/private/myfile"
@DTWF_REMOVE(myFile)
```

Example 2:

```
%DEFINE {
  myFile = "/private/myfile"
  myWait = "2000"
%}
@DTWF_REMOVE(myFile, myWait)
```

DTWF_SEARCH

Purpose

Searches a text file for a string, returning the results into a Net.Data table.

Format

@DTWF_SEARCH(filename, transform, delimiter, table, searchFor, retry, linesToSearch, startline)
@DTWF_SEARCH(filename, transform, delimiter, table, searchFor, retry, linesToSearch)
@DTWF_SEARCH(filename, transform, delimiter, table, searchFor, retry)
@DTWF_SEARCH(filename, transform, delimiter, table, searchFor)

Parameters

Table 189. 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. A new-line character in a file indicates the end of a row of a Net.Data macro table for ASCIITEXT and DELIMITED transforms.
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: • The line in which the match was found • The field in which the match was found (for ASCIITEXT transform, this is always 1) • The value of the field that contained the search string
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	*linesToSearch*	INOUT	The number of lines in the file to search. A value of 0 means all the lines in the file are searched or until the table is full; this is the default. Upon successful completion, the number of rows in the resulting table is returned by this parameter.
integer	*startline*	IN	The line in the file to start searching from. The default is 1, which begins the search at the first record.

Return codes

Table 190. DTWF_SEARCH Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
1010	Data was written to the table until it was full, and the remainder of the data was discarded.
2000	A flat file interface built-in function could not find the specified file.
2003	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Usage Notes

1. The table that is returned for DTWF_SEARCH has three columns. The first two columns contain the row and the column number where the match is found; the last column contains the column value that contains the characters that are specified in the *SearchFor* parameter. For example, if the fourth row of the file contains matching characters in column three, the returned table has a row with the number 4 in the first column to indicate the row of the file that it came from; it has a number 3 in the second column to indicate which column of the file contains a match; and it has the complete column value in the third column.
2. The SearchFor parameter cannot include the contents of the *delimiter* parameter.

Examples

Example 1:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
  myWait = "1000"
  mySearch = "0123456789abcdef"
@DTWF_SEARCH(myFile, "DELIMITED", ";",
               myTable, mySearch, myWait)
```

Example 2:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
  mySearch = "answer:"
  myWait = "0"
  myRows = "0"
  myStartrow = "1"
%}
@DTWF_SEARCH(myFile, "DELIMITED", ";", myTable,
               mySearch, myWait, myRows, myStartrow)
```

DTWF_UPDATE

Purpose

Update lines in a text file with data from a Net.Data table.

Format

@DTWF_UPDATE(filename, transform, delimiter, table, retry, rows, startline)
@DTWF_UPDATE(filename, transform, delimiter, table, retry, rows)
@DTWF_UPDATE(filename, transform, delimiter, table, retry)
@DTWF_UPDATE(filename, transform, delimiter, table)

Parameters

Table 191. 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. A new-line character in a file indicates the end of a row of a Net.Data macro table for ASCIITEXT and DELIMITED transforms.
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 number of rows in the table to use when updating the file. A value of 0 means to use all the rows when updating the file; this is the default. Note that only existing lines in the file are updated, no lines are added.
integer	*startline*	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 lines in a file, the value is changed to indicate the number of the last line in the file and an error is returned.

Return codes

Table 192. DTWF_UPDATE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.

Table 192. DTWF_UPDATE Return Codes (continued)

Return Code	Explanation
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2001	A flat file interface built-in function could not open the specified file because it was in use by this or another process, and could not be shared in the specified mode.
2003	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Usage Notes

If the file does not exist, an absolute path for the filename should be specified, and the directory where the file is to be created must match a directory specified in FFI_PATH. If an absolute path is not used, the file will be opened in the current working directory.

Examples

Example 1:

```
%DEFINE {
   myFile = "/private/myfile"
   myTable = %TABLE
   myWait = "1500"
   myRows = "2"
%}
@DTWF_UPDATE(myFile, "Delimited", "|", myTable, myWait, myRows)
```

Example 2:

```
%DEFINE {
   myFile = "/private/myfile"
   myTable = %TABLE
   myStart = "1"
   myRows = "2"
%}
@DTWF_UPDATE(myFile, "Asciitext", "|", myTable, "0", myRows, myStart)
```

DTWF_WRITE

Purpose

Writes the contents of a Net.Data table to a text file.

Format

@DTWF_WRITE(filename, transform, delimiter, table, retry, rows, startline)
@DTWF_WRITE(filename, transform, delimiter, table, retry, rows)
@DTWF_WRITE(filename, transform, delimiter, table, retry)
@DTWF_WRITE(filename, transform, delimiter, table)

Parameters

Table 193. 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. A new-line character in a file indicates the end of a row of a Net.Data macro table for ASCIITEXT and DELIMITED transforms.
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 number of rows in table to write into the file. A value of 0 means that all rows in table are written into the file; this is the default.
integer	*startline*	INOUT	The line number in the file from which to begin writing. A value of 1 means to start at the first line in the file; this is the default. If a value beyond the end of the file is specified, an error is returned and this parameter is set to the number of lines in the file.

Return codes

Table 194. DTWF_WRITE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.

Table 194. DTWF_WRITE Return Codes (continued)

Return Code	Explanation
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2003	A flat file interface built-in function could not read a row of data into a table variable because the number of bytes in the row exceeded the maximum supported number of bytes.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Usage Notes

- If the file does not exist, an absolute path for the filename should be specified, and the directory where the file is to be created must match a directory specified in FFI_PATH.
- If an absolute path is not used, the file will be opened in the current working directory.
- If a file has not been previously opened, DTW_WRITE() will clear the file of all of its contents before writing the data from the passed-in table to the file.

Examples

Example 1:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
%}
@DTWF_WRITE(myFile, "DELIMITED", ";", myTable)
```

Example 2:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
%}
@DTWF_WRITE(myFile, "ASCIITEXT", ";", myTable, "5000")
```

Example 3:

```
%DEFINE {
  myFile = "/private/myfile"
  myTable = %TABLE
%}
@DTWF_WRITE(myFile, "ASCIITEXT", ";", myTable, "5000", "10", "50")
```

DTWF_WRITEFILE

Purpose

Writes a string to a file.

Format

@DTWF_WRITEFILE(filename, dataString, appendOptions)
@DTWF_WRITEFILE(filename, dataString)

Parameters

Table 195. DTW_WRITEFILE Parameters

Data Type	Parameter	Use	Description
string	*filename*	INOUT	The name of the file that the data specified by *dataString* is written to. On successful completion of the call
string	*dataString*	IN	The variable that is assigned the contents of the file.
string	*appendOptions*	IN	The variable that is assigned the contents of the file.

Return codes

Table 196. DTWF_WRITEFILE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
2000	A flat file interface built-in function could not find the specified file.
2001	A flat file interface built-in function could not open the specified file because it was in use by this or another process, and could not be shared in the specified mode.
2004	A flat file interface built-in function was attempting to find a file, but encountered a path in the FFI_PATH configuration file variable that was longer than the maximum supported number of bytes, which is 4095.
2005	A call to a system function failed.
2006	A flat file interface built-in function could not access the specified file because it was in use by this or another process and could not be shared in the specified mode.

Usage Notes

- If the file does not exist, specify an absolute path for the filename. The directory where the file is to be created must match a directory specified in the FFI_PATH. If an absolute path is not used, the file will be opened in the current working directory.

- If the specified file has been opened using DTWF_OPEN with an open mode of "w" and if the append option is set to "N," then DTWF_WRITEFILE will clear any data in the file.

Examples

Example 1:

```
%DEFINE {
myFile = "/private/myfile"
mymsg = "This is a write file test1!"
%}
@DTWF_WRITEFILE(myFile, mymsg)
```

Example 2:

```
%DEFINE {
myFile = "/private/myfile"
mymsg = "This is a write file test1!"
%}
@DTWF_WRITEFILE(myFile, mymsg, "Y")
```

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.

On the OS/400 platform, registries are implemented using physical files.

DTWR_ADDENTRY

Purpose

Adds an entry to a Web registry.

Format

@DTWR_ADDENTRY(registry, registryVariable, registryData, index)
@DTWR_ADDENTRY(registry, registryVariable, registryData)

Parameters

Table 197. 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.

Return codes

Table 198. DTWR_ADDENTRY Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3003	A Web registry built-in function could not add an entry to the specified registry because the specified entry already exists.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3006	A Web registry built-in function could not create the specified registry because a path in the registry name does not exist.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Examples

Example 1:

```
@DTWR_ADDENTRY("/qsys.lib/mylib.lib/myreg.file",
               "Jones", "http://Advantis.com/~Jones/webproj")
```

Example 2:

```
@DTWR_ADDENTRY("/qsys.lib/mylib.lib/urllist.file",
               "SMITH", "http://www.ibm.com/software/",
               "WORK_URL,")
```

DTWR_CLEARREG

Purpose

Clears entries from a Web registry.

Format

@DTWR_CLEARREG(registry)

Parameters

Table 199. DTWR_CLEARREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to clear.

Return codes

Table 200. DTWR_CLEARREG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3006	A Web registry built-in function could not create the specified registry because a path in the registry name does not exist.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Examples

Example 1:

```
@DTWR_CLEARREG("/qsys.lib/mylib.lib/myreg.file")
```

DTWR_CLOSEREG

Purpose

Closes a Web registry

Format

@DTWR_CLOSEREG(registry)

Parameters

Table 201. DTWR_CLOSEREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to close. **Restriction:** Do not use special characters such as the asterisk (*) and the backslash (\) in Web registry names.

Return codes

Table 202. DTWR_CLOSEREG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.

Examples

Example 1: Close a registry

```
@DTWR_CLOSEREG("/qsys.lib/mylib.lib/myreg.file")
```

DTWR_CREATEREG

Purpose

Creates a new Web registry.

Format

@DTWR_CREATEREG(registry, security)
@DTWR_CREATEREG(registry)

Parameters

Table 203. DTWR_CREATEREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to create. The registry name can be a fully qualified integrated file system name such as "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE"; or the registry name can be just the filename, "MYFILE.FILE", with the current working directory used for the library.
string	*security*	IN	The type of security with which to create *registry*. The security parm may either be a UNIX-style parameter or an AS400 security parameter. The UNIX-style parm is detected by 2 commas "**,**" in the string. For example "**,,**" grants no authority to the 3 unix security groups, user, group, and public. A value of "*RWX,*RWX,*RWX" grants read, write, and execute to all 3 groups. The AS400 security parm may not contain leading, trailing, or embedded blanks. Allowed values are: *CHANGE, *EXCLUDE, *LIBCRTAUT, *ALL, *USE, or a valid AS400 authorization list name.The security parm is optional. If not specified, the AS400 security value *LIBCRTAUT is used to generate the registry.

Return codes

Table 204. DTWR_CREATEREG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3002	A Web registry built-in function could not delete the specified registry.
3006	A Web registry built-in function could not create the specified registry because a path in the registry name does not exist.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Table 204. DTWR_CREATEREG Return Codes (continued)

Return Code	Explanation
3008	A Web registry built-in function could not create the specified registry for unknown reasons.

Examples

Example 1:

```
@DTWR_CREATEREG("/qsys.lib/mylib.lib/myreg.file")
```

Example 2:

```
@DTWR_CREATEREG("/qsys.lib/mylib.lib/urllist.file", "*RWX, *RWX, *R")
```

DTWR_DELENTRY

Purpose

Deletes an entry from a Web registry.

Format

@DTWR_DELENTRY(registry, registryVariable, index)
@DTWR_DELENTRY(registry, registryVariable)

Parameters

Table 205. 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.

Return codes

Table 206. DTWF_DELENTRY Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3004	A Web registry built-in function could not remove or retrieve an entry from the specified registry because the specified entry does not exist.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Examples

Example 1:

```
@DTWR_DELENTRY("/qsys.lib/mylib.lib/myreg.file", "Jones")
```

Example 2:

```
@DTWR_DELENTRY("/qsys.lib/mylib.lib/urllist.file",
               "SMITH", "WORK_URL")
```

DTWR_DELREG

Purpose

Deletes a Web registry

Format

@DTWR_DELREG(registry)

Parameters

Table 207. DTWR_DELREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to delete.

Return codes

Table 208. DTWR_DELREG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Examples

Example 1:

```
@DTWR_DELREG("/qsys.lib/mylib.lib/myreg.file")
```

DTWR_LISTREG

Purpose

Lists the contents of a Web registry.

Format

@DTWR_LISTREG(registry, registryTable)

Parameters

Table 209. DTWR_LISTREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to list.
table	*registryTable*	OUT	The name of the table variable in which the registry entries are placed.

Return codes

Table 210. DTWR_LISTREG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1004	A parameter passed on a function call, required to be a Net.Data macro table variable, was of a different variable type.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Usage Notes

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.

Examples

Example 1:

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

@DTWR_LISTREG("/qsys.lib/mylib.lib/urllist.file", RegistryTable)
```

DTWR_OPENREG

Purpose

Opens a Web registry.

Format

@DTWR_OPENREG(registry, commit)
@DTWR_OPENREG(registry)

Parameters

Table 211. DTWR_OPENREG Parameters

Data Type	Parameter	Use	Description
string	*registry*	IN	The name of the registry to open.
string	*commit*	IN	A single symbol or literal string specifying whether the registry is opened under commitment control or not. The possible values are: **Y** Open the registry under commitment control. **N** Do not open the registry under commitment control. The default is N

Return codes

Table 212. DTWR_OPENREG Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Examples

Example 1: Open the registry under commitment control

```
@DTWR_OPENREG("/qsys.lib/mylib.lib/myreg.file", "Y")
```

DTWR_RTVENTRY

Purpose

Retrieves a registry string value from a Web registry.

Format

@DTWR_RTVENTRY(registry, registryVariable, registryData, index)
@DTWR_RTVENTRY(registry, registryVariable, registryData)
@DTWR_rRTVENTRY(registry, registryVariable, index)
@DTWR_rRTVENTRY(registry, registryVariable)

Parameters

Table 213. 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.

Return codes

Table 214. DTWR_RTVENTRY Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.
1007	A parameter contains a value which is not valid.
3004	A Web registry built-in function could not remove or retrieve an entry from the specified registry because the specified entry does not exist.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Examples

Example 1:

```
%DEFINE RegistryData = ""
@DTWR_RTVENTRY("/qsys.lib/mylib.lib/myreg.file", "Jones", RegistryData)
```

Example 2:

```
@DTWR_RTVENTRY("/qsys.lib/mylib.lib/urllist.file",
               "SMITH", RegistryData, "WORK_URL")
```

Example 3:

```
@DTWR_rRTVENTRY("/qsys.lib/mylib.lib/myreg.file", "Jones")
```

Example 4:

```
@DTWR_rRTVENTRY("/qsys.lib/mylib.lib/urllist.file",
                "SMITH", "WORK_URL")
```

DTWR_UPDATEENTRY

Purpose

Updates a registry string value in the Web registry.

Format

@DTWR_UPDATEENTRY(registry, registryVariable, newData, index)
@DTWR_UPDATEENTRY(registry, registryVariable, newData)

Parameters

Table 215. 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.

Return codes

Table 216. DTWR_UPDATEENTRY Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1002	An input parameter contained a string value which consisted of the null-terminating character.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
3003	A Web registry built-in function could not add an entry to the specified registry because the specified entry already exists.
3004	A Web registry built-in function could not remove or retrieve an entry from the specified registry because the specified entry does not exist.
3005	A Web registry built-in function could not use the specified registry because it cannot be found.
3007	A Web registry built-in function could not complete the specified operation because the requestor does not have the proper authority to the specified registry.

Usage Notes

The registry entry name corresponding to the value cannot be changed.

Examples

Example 1:

```
@DTWR_UPDATEENTRY("/qsys.lib/mylib.lib/myreg.file",
                  "Jones", "http://advantis.com/~Jones/personal")
```

Example 2:

```
@DTWR_UPDATEENTRY("/qsys.lib/mylib.lib/urllist.file",
                  "SMITH", "http://www.ibm.com/software/personal",
                  "WORK_URL")
```

Persistent macro functions

The persistent macro functions support transaction processing in Net.Data by helping you define which macro blocks are persistent within a single transaction. Use these functions to define the start and end of a transaction, which HTML blocks are persistent throughout the transaction, the scope of the variables within the transaction, and whether to commit or rollback changes within the transaction.

DTW_ACCEPT

Purpose

Defines the transaction handle used to invoke a persistent macro.

Format

@DTW_ACCEPT(handle, timeout)

@DTW_ACCEPT(handle)

Parameters

Table 217. DTW_ACCEPT Parameters

Data Type	Parameter	Use	Description
string	*handle*	IN	A variable or literal string specifying a transaction handle to be used in URLs for subsequent macro invocations in this persistent transaction.
integer	*timeout*	IN	A variable or literal string specifying an amount of time in seconds for the job servicing this port to wait for a response. This value overrides any timeout value specified on the DTW_STATIC() function.

Return codes

Table 218. DTW_ACCEPT Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
8200	Macro persistence is not enabled.
8201	A persistent built-in function was called out of sequence.

Usage Notes

1. Net.Data requires that the transaction handle be included in the URL that invokes the macro as a response from the Web browser. When a request comes in to the Web server, the server uses the transaction handle to route the request to the CGI process that is processing the transaction.

   The transaction handle must be called at the start of each HTML block in the macro until the last logical block, which contains a call to DTW_TERMINATE(). If either a call to DTW_ACCEPT() or DTW_TERMINATE() is not found before any text is output to the browser, a Net.Data error occurs.
2. You can specify a timeout value for this page that overrides the timeout value specified on the @DTW_STATIC() function. The Web server waits for specified amount of time (in seconds) for the user to respond to this request.
3. If this function is called when the macro is not in a persistent state, a Net.Data error occurs.
4. The URLs containing the transaction handle can be coded as actions on form push buttons or as hypertext links on the page presented to the browser.

Examples

Example 1:

```
%DEFINE handle = ""
@DTW_RTVHANLDE(handle)

%HTML (Report){
@DTW_ACCEPT(handle)
 ...
%}
```

DTW_COMMIT

Purpose

Makes permanent any pending changes made to resources under commitment control since the last commitment boundary and establishes a new commitment boundary.

Format

@DTW_COMMIT()

Parameters

None.

Return codes

Table 219. DTW_COMMIT Return Codes

Return Code	Explanation
1003	An incorrect number of parameters were passed on a function call.

Examples

Example 1: Specifies a commit

```
@DTW_COMMIT()
%HTML (Report){
%}
```

DTW_ROLLBACK

Purpose

Reestablishes the last commitment boundary as the current commitment boundary. All changes to resources under commitment control for the process that Net.Data is running under made since the last commitment boundary are backed out.

Format

@DTW_ROLLBACK()

Parameters

None.

Return codes

Table 220. DTW_ROLLBACK Return Codes

Return Code	Explanation
1003	An incorrect number of parameters were passed on a function call.

Examples

Example 1: Specifies a rollback

```
@DTW_ROLLBACK()
%HTML (Report){
%}
```

DTW_RTVHANDLE

Purpose

Generates and returns a transaction handle that is unique to this macro across separate invocations and is calculated based on a combination of thread information, timestamp, and current user.

Format

@DTW_RTVHANDLE(handle)

Parameters

Table 221. DTW_RTVHANDLE Parameters

Data Type	Parameter	Use	Description
string	*handle*	OUT	A variable that contains a unique transaction handle for the current persistent macro.

Return codes

Table 222. DTW_RTVHANDLE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1006	A literal string was passed on a function call for a parameter which was required to be an output parameter.

Usage Notes

The transaction handle can be used to ensure that URLs specified as part of a persistent transaction are unique to the HTTP server and can be securely identified as valid requests.

Examples

Example 1: Defines the `handle` variable used to retrieve the transaction handle

```
%DEFINE handle = ""
@DTW_RTVHANLDE(handle)
```

DTW_STATIC

Purpose

Indicates that the entire macro is persistent.

Format

@DTW_STATIC(timeout)

@DTW_STATIC()

Parameters

Table 223. DTW_STATIC Parameters

Data Type	Parameter	Use	Description
integer	*timeout*	IN	A variable or literal string that specifies an amount of time, in seconds, that the process handling this transaction should wait for a response.

Return codes

Table 224. DTW_STATIC Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1001	An input parameter contained a NULL value.
1003	An incorrect number of parameters were passed on a function call.
1005	A parameter passed on a function call, required to be a string variable, was of a different variable type.
1007	A parameter contains a value which is not valid.
8202	Persistence could not be enabled.

Usage Notes

1. DTW_STATIC should be the first statement in the macro. All variables defined in the macro after this function call will be persistent across multiple macro invocations unless specified otherwise and until DTW_TERMINATE() is called or the process is ended.
2. A timeout value, in seconds, can be specified on the function call to indicate the amount of time the process Net.Data is running under waits for a response from the browser. If the timeout value expires, the process ends, and all changes to resources under commitment control since the last commitment boundary are rolled back.
3. If a timeout value is specified on a subsequent @DTW_ACCEPT() call, Net.Data overrides this value with the value in the subsequent call. If a timeout value is not specified on this call or a subsequent @DTW_ACCEPT() call, the Web server default timeout value is used.

Examples

Example 1: A call to DTW_STATIC() specifying a timeout value of 60 seconds.

```
@DTW_STATIC("60")
```

DTW_TERMINATE

Purpose

Ends a persistent transaction. All changes to resources under commitment control since the last commitment boundary are made permanent.

Format

@DTW_TERMINATE()

Parameters

None

Return codes

Table 225. DTW_TERMINATE Return Codes

Return Code	Explanation
-1001	The server could not process a Net.Data request to allocate memory.
1003	An incorrect number of parameters were passed on a function call.
8200	Macro persistence is not enabled.
8201	A persistent built-in function was called out of sequence.

Usage Notes

1. The DTW_TERMINATE function is called at the start of the logical last HTML block of the persistent transaction before any text is output to the browser. If any text output appears before the function, within the block, a Net.Data error will occur. Note that there could be more than one logical last HTML block depending on how the application is written.
2. If this function is called when the macro is not in a persistent state, a Net.Data error will occur.

Examples

Example 1: Terminates the persistent transaction

```
%HTML(QUIT){
@DTW_TERMINATE()
 ...
%}
```

Java applet functions

The Java applet support lets you easily generate HTML tags for Java applets in your Net.Data applications. When you call the Java applet functions, you specify the name of your applet and pass any parameters that the applet needs. The language environment processes the macro and generates the HTML applet tags, which the Web browser uses to run the applet.

The following sections describe how to use the Java applet language environment to run your Java applets.

Creating Java applets

Before using the Net.Data Java applet functions, you need to determine which applets you plan to use or which applets you need to write. See your Java documentation for more information on creating applets.

Generating the applet tags

You specify a call to the Net.Data applet support with a Net.Data function call. No declaration is needed for the function call. The syntax for the function call is shown here:

```
@DTWA_AppletName(parm1, parm2, ..., parmN)
```

- DTWA_ identifies the applet function call.
- AppletName is the name of the applet for which tags are generated.
- parm1 through parmN are parameters used to generate PARAM tags.

To write a macro that generates applet tags:

1. Define any parameters required by the applet in the DEFINE section of the macro. These parameters include any applet tag attributes, Net.Data variables, and Net.Data table parameters that you need as input for the applet. For example:

   ```
   %define{
   DATABASE = "celdial"                    <=Name of the database
   MyGraph.codebase = "/netdata-java/"     <=Required applet attribute
   MyGraph.height = "200"                  <=Required applet attribute
   MyGraph.width = "400"                   <=Required applet attribute
   MyTitle = "Celdial results"             <=Name of the Web page
   MyTable = %TABLE(all)                   <=Table to store query results
   %}
   ```

2. Optional: Specify a query to the database to generate a result set as input for the applet. This is useful when you are using an applet that generates a chart or table. For example:

   ```
   %FUNCTION(DTW_SQL) mySQL(OUT table){
   select name, ages from ibmuser.guests
   %}
   ```

3. Specify the function call in the Net.Data macro to call the Java applet support. The function call specifies the name of the applet and the parameters you want to pass. These parameters include any Net.Data variables, and Net.Data table or column parameters that you need as input for the applet.

   For example:

   ```
   %HTML (Report){
   @mySQL(MyTable)                                  <=A call to mySQL
   @DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable)  <=Applet function call
   %}
   ```

Applet tag attributes

You can specify attributes for applet tags anywhere in your Net.Data macro. Net.Data substitutes all variables that have the form *AppletName.attribute* into the applet tag as attributes. The syntax for defining an attribute on an applet tag is shown here:

```
%define AppletName.attribute = "value"
```

The following attributes are required for all applets:

- *codebase:* The location of the applet, which is identified by a URL.
- *height:* The height of the applet in pixels.
- *width:* The width of the applet in pixels.

The following attributes are optional:

- *align:* the alignment of the applet
- *alt:* any text that should be displayed if the browser understands the APPLET tag but can't run Java applets
- *archive:* an archive containing classes and other resources
- *hspace:* the number of pixels on each side of the applet
- *name:* a name for the applet instance
- *object:* the name of the file that contains a serialized representation of an applet
- *vspace:* the number of pixels above and below the applet

For example, if your applet is called MyGraph, you can define these required attributes as shown here:

```
%DEFINE{
MyGraph.codebase = "/netdata-java/"
MyGraph.height   = "200"
MyGraph.width    = "400"
%}
```

The actual assignment need not be in a DEFINE section. You can set the value with the DTW_ASSIGN function. If you do not define a variable for *AppletName.code* variable, Net.Data adds a default *code* parameter to the applet tag. The value of the *code* parameter is *AppletName.class*, where *AppletName* is the name of your applet.

Applet tag parameters

You define a list of parameters to pass to the Java applet in the function call. You can pass parameters that include:

- Net.Data variables (including LIST variables)
- Net.Data tables
- Columns of Net.Data tables

When you pass a parameter, Net.Data creates a Java applet PARAM tag in the HTML output with the name and value that you assign to the parameter. You cannot pass string literals or results of function calls.

Net.Data variable parameters:

You can use Net.Data variables as parameters. If you define a variable in the DEFINE block of the macro and pass the variable value in the DTWA_*AppletName* function call, Net.Data generates a PARAM tag that has the same name and value as the variable. For example, given the following macro statement:

```
%define{

...

MyTitle = "This is my Title"
%}

%HTML (Report){
@DTWA_MyGraph( MyTitle, ...)
%}
```

Net.Data produces the following applet PARAM tag:

```
<param name = 'MyTitle' value = "This is my Title" />
```

Net.Data table parameters:

Net.Data automatically generates a PARAM tag with the name DTW_NUMBER_OF_TABLES every time the Java applet function is called, specifying whether the function call has passed any table variables. The value is the number of table variables that Net.Data uses in the function. If no table variables are specified in the function call, the following tag is generated:

```
<param name = "DTW_NUMBER_OF_TABLES" value = "0" />
```

You can pass one or more Net.Data table variables as parameters on the function call. If you specify a Net.Data table variable on a DTWA_*AppletName* function call, Net.Data generates the following PARAM tags:

Table name parameter tag:

This tag specifies the names of the tables to pass. The tag has the following syntax:

```
<param name = 'DTW_TABLE_i_NAME' value = "tname" />
```

Where `i` is the number of the table based on the ordering of the function call, and *tname* is the name of the table.

Row and column specification parameter tags:

PARAM tags are generated to specify the number of rows and columns a particular table. This tag has the following syntax:

```
<param name = 'DTW_tname_NUMBER_OF_ROWS' value = "rows" />
<param name = 'DTW_tname_NUMBER_OF_COLUMNS' value = "cols" />
```

Where the name of the table is *tname*, *rows* is the number of rows in the table, and *cols* is the number of columns in the table. This pair of tags is generated for each unique table specified in the function call.

Column value parameter tags:

This PARAM tag specifies the column name of a particular column. This tag has the following syntax:

```
<param name = 'DTW_tname_COLUMN_NAME_j' value = "cname" />
```

Where the table name is *tname*, *j* is the column number, and *cname* is the name of the column in the table.

Row value parameter tags:

This PARAM tag specifies the values at a particular row and column. This tag has the following syntax:

```
<param name = 'DTW_tname_cname_VALUE_k' value = "val" />
```

Where the table name is *tname*, *cname* is the column name, *k* is the row number, and *val* is the value that matches the value in the corresponding row and column.

Table column parameters: You can pass a table column as a parameter on a function call to generate tags for a specific column. Net.Data generates the corresponding applet tags only for the specified column. A table column parameter uses the following syntax:

```
@DTWA_AppletName(DTW_COLUMN( x )Table)
```

Where *x* is the name or number of the column in the table.

Table column parameters use the same applet tags defined for the table parameters.

Alternate text for the applet tag on browsers that are not Java-enabled

The variable DTW_APPLET_ALTTEXT specifies the text to display on browsers that do no support Java or have turned Java support off. For example, the following variable definition:

```
%define DTW_APPLET_ALTTEXT = "<p>Sorry, your browser is not Java-enabled."</p>
```

produces the following HTML tag and text:

```
<p>Sorry, your browser is not Java-enabled.</p><
```

If this variable is not defined, no alternate text is displayed.

Java applet example

The following example demonstrates a Net.Data macro that calls the Java applet support and the resulting applet tag that is generated.

The Net.Data macro contains the following function calls to the Java applet support:

```
%define{
DATABASE = "celdial"
DTW_APPLET_ALTTEXT = "<p>Sorry, your browser is not Java-enabled."</p>
DTW_DEFAULT_REPORT = "no"
MyGraph.codebase = "/netdata-java/"
MyGraph.height = "200"
MyGraph.width = "400"
MyTitle = "This is my Title"
%}
%FUNCTION(DTW_SQL) mySQL(OUT table){
select name, ages from ibmuser.guests
%}
%HTML (Report){
@mySQL(MyTable)
@DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable)
%}
```

The Net.Data macro lines in the DEFINE section specify the attributes of the applet tag:

```
MyGraph.codebase = "/netdata-java/"
MyGraph.height = "200"
MyGraph.width = "400"
```

The Java applet support generates an applet tag with the following qualifiers:

```
<applet code='MyGraph.class'
        codebase='/netdata-java/'
 width='400'
 height='200'
>
```

Net.Data returns the SQL query results from the SQL section of the Net.Data macro in the output table, `MyTable`. This table is specified in the DEFINE section:

```
MyTable = %TABLE(all)
```

The call to the applet in the macro is specified in the HTML section:

```
@DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable)
```

Based on the parameters in the function call, Net.Data generates the complete applet tag containing the information about the result table, such as the number of columns, the number of rows returned, and the result rows. Net.Data generates one parameter tag for each cell in the result table, as shown in the following example:

```
<param name = 'DTW_MyTable_ages_VALUE_1' value = "35" />
```

The parameter name, *DTW_MyTable_ages_VALUE_1*, specifies the table cell (row 1, column ages) in the table, MyTable, which has a value of 4. The keyword, DTW_COLUMN, in the function call to the applet, specifies that you are interested only in the column ages of the resulting table, MyTable, shown here:

```
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
```

The following output shows the complete applet tag that Net.Data generates for the example:

```
<applet code='MyGraph.class' codebase='/netdata-java/'
        width='400'          height='200'
>
<param name = 'MyTitle' value = "This is my Title" />
<param name = 'DTW_NUMBER_OF_TABLES' value = "1" />
<param name = 'DTW_TABLE_1_NAME' value = "MyTable" />
<param name = 'DTW_MyTable_NUMBER_OF_ROWS' value = "5" />
<param name = 'DTW_MyTable_NUMBER_OF_COLUMNS' value = "1" />
<param name = 'DTW_MyTable_COLUMN_NAME_1' value = "ages" />
<param name = 'DTW_MyTable_ages_VALUE_1' value = "35" />
<param name = 'DTW_MyTable_ages_VALUE_2' value = "32" />
<param name = 'DTW_MyTable_ages_VALUE_3' value = "31" />
<param name = 'DTW_MyTable_ages_VALUE_4' value = "28" />
<param name = 'DTW_MyTable_ages_VALUE_5' value = "40" />
<p>Sorry, your browser is not Java-enabled.</p>
</applet>
```

Appendix A. Net.Data technical library

The Net.Data technical library is available from the Net.Data Web site at http://www.ibm.com/systems/i/software/netdata/docs.

Document	Description
Net.Data Administration and Programming Guide	Contains conceptual and task information about installing, configuring, and invoking Net.Data. Also describes how to write Net.Data macros, use Net.Data performance techniques, use Net.Data language environments, manage connections, and use Net.Data logging and traces for trouble shooting and performance tuning.
Net.Data Reference	Describes the Net.Data macro language, variables, and built-in functions.
Net.Data Language Environment Interface Reference	Describes the Net.Data language environment interface.
Net.Data Messages and Return Codes	Lists Net.Data error messages and return codes.

Related documentation

The following documents might be useful when using Net.Data and related products:

- *DB2 for IBM i SQL Programming*
- *IBM i Distributed Database Programming*

Additionally, OS/400 documentation and redbooks, including books about DB2, are available at the following URL:

`http://publib.boulder.ibm.com/iseries/`

Appendix B. Deprecated features

The following features are still supported, but not recommended. If you have macros that contain these language constructs, it is recommended that you use the suggested alternatives.

EXEC_SQL

A DB2 WWW Connection language construct that calls an SQL block. We recommend calling SQL statements as functions instead. See "FUNCTION block" on page 16 for more information.

HTML_INPUT

A DB2 WWW Connection language construct that is the same as an HTML block named INPUT. See "HTML block" on page 24 for more information.

HTML_REPORT

A DB2 WWW Connection language construct that is the same as an HTML block named REPORT. See "HTML block" on page 24 for more information.

INCLUDE_URL

A statement used to read and incorporate another file into the Net.Data generated output. Upon execution, the entire contents of the included file will replace the INCLUDE_URL statement. The specified file can exist on a local or remote server.

Though there is no recommended alternative to including the contents of a remote URL, you can use the INCLUDE statement to include local files, including macros. Including remote URLs is not recommended because there is no way to trap any of the unexpected situations that might occur when accessing a remote site.

NUM_ROWS

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 49 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.

You can reference NUM_ROWS in REPORT and ROW blocks.

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

```
%DEFINE DTW_SET_TOTAL_ROWS="YES"
%DEFINE RPT_MAX_ROWS="10"

%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)
%}
```

SQL

A DB2 WWW Connection language construct that 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 2: 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

A DB2 WWW Connection language construct that is equivalent to the Net.Data MESSAGE statement. See “MESSAGE block” on page 40 for an example.

SQL_REPORT

A DB2 WWW Connection language construct that is equivalent to the Net.Data REPORT statement. See “REPORT block” on page 44for an example.

SQL_CODE

A DB2 WWW Connection language construct that is equivalent to “RETURN_CODE” on page 100.

Appendix C. Net.Data supported features by operating system

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.

Table 226. Net.Data Language Environments

Language Environment	AIX	HP	Linux (x386 & S/390)	OS/2	OS/390	OS/400	SUN	Win NT
Direct Call						X		
Java Applications	X			X		X	X	X
REXX	X		X	X	X	X	X	X
SQL	X	X	X	X	X	X	X	X
System	X	X	X	X	X	X	X	X

Table 227. Net.Data Configuration Variables

Configuration Variable	AIX	HP	Linux (x386 & S/390)	OS/2	OS/390	OS/400	SUN	Win NT
DTW_ATTACHMENT_PATH	X	X	X	X		X	X	X
DTW_DEFAULT_ERROR_MESSAGE	X	X	X	X	X	X	X	X
DTW_DEFAULT_MACRO					X	X		
DTW_ERROR_LOG_DIR					X	X		
DTW_ERROR_LOG_LEVEL					X	X		
DTW_LOB_DIR					X	X		
DTW_REMOVE_WS	X	X	X	X	X	X	X	X
DTW_SHOWSQL	X	X	X	X	X	X	X	X
DTW_SMTP_CHARSET						X		
DTW_SMTP_SERVER	X	X	X	X		X	X	X
DTW_SQL_ISOLATION						X		
DTW_SQL_NAMING_MODE						X		
DTW_TRACE_LOG_DIR	X		X		X	X	X	X
DTW_TRACE_LOG_LEVEL	X		X		X	X	X	X
DTW_UPLOAD_DIR	X	X	X	X	X	X	X	X
DTW_VARIABLE_SCOPE	X	X	X	X		X	X	X
EXEC_PATH	X	X	X	X	X	X	X	X
FFI_PATH	X	X	X	X	X	X	X	X
HTML_PATH	X	X	X	X	X	X	X	X
INCLUDE_PATH	X	X	X	X	X	X	X	X
MACRO_PATH	X	X	X	X	X	X	X	X

Table 228. Net.Data Variables

Variable	AIX	HP	Linux (x386 & S/390)	OS/2	OS/390	OS/400	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
DTW_APPLET_ALTTEXT	X	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	X	X	X	X	X	X	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_MACRO_FILENAME	X	X	X	X	X	X	X	X
DTW_MACRO_LAST_MODIFIED	X	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_PAD_PGM_PARMS						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	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	X
TOTAL_ROWS	X	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 229. Net.Data Functions

Function	AIX	HP	Linux (x386 & S/390)	OS/2	OS/390	OS/400	SUN	Win NT
DTW_ACCEPT						X		
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_CHARTOHEX	X	X	X	X	X	X	X	X
DTW_COMMIT						X		
DTW_CONCAT	X	X	X	X	X	X	X	X
DTW_DATATYPE					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
DTW_DIVREM	X	X	X	X	X	X	X	X
DTW_EXIT	X	X	X	X	X	X	X	X
DTW_EVAL						X		
DTW_FORMAT	X	X	X	X	X	X	X	X
DTW_GETCOOKIE	X	X	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_HEXTOCHAR	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_ISNUMERIC					X	X		
DTW_LASTPOS	X	X	X	X	X	X	X	X
DTW_LENGTH	X	X	X	X	X	X	X	X
DTW_LOG_ERRORMSG					X	X		
DTW_LOG_TRACEMSG					X	X		
DTW_LOWERCASE	X	X	X	X	X	X	X	X
DTW_MULTIPLY	X	X	X	X	X	X	X	X
DTW_PAD						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_REPLACE	X	X	X	X	X	X	X	X
DTW_REVERSE	X	X	X	X	X	X	X	X
DTW_ROLLBACK						X		
DTW_RVTHANDLE						X		

Table 229. Net.Data Functions (continued)

Function	AIX	HP	Linux (x386 & S/390)	OS/2	OS/390	OS/400	SUN	Win NT
DTW_SENDMAIL	X	X	X	X	X	X	X	X
DTW_SETCOOKIE	X	X	X	X	X	X	X	X
DTW_SETENV	X	X	X	X	X	X	X	X
DTW_STATIC						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_APPENDROW	X	X	X	X	X	X	X	X
DTW_TB_COLS	X	X	X	X	X	X	X	X
DTW_TB_DELETECOL	X	X	X	X	X	X	X	X
DTW_TB_DELETEROW	X	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	X
DTW_TB_GETV	X	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_INSERTCOL	X	X	X	X	X	X	X	X
DTW_TB_INSERTROW	X	X	X	X	X	X	X	X
DTW_TB_LIST	X	X	X	X	X	X	X	X
DTW_TB_MAXROWS	X	X	X	X		X	X	X
DTW_TB_QUERYCOLNONJ	X	X	X	X	X	X	X	X
DTW_TB_ROWS	X	X	X	X	X	X	X	X
DTW_TB_SELECT	X	X	X	X	X	X	X	X
DTW_TB_SETCOLS	X	X	X	X	X	X	X	X
DTW_TB_SETN	X	X	X	X	X	X	X	X
DTW_TB_SETV	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_TERMINATE						X		
DTW_TIME	X	X	X	X	X	X	X	X
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

Table 229. Net.Data Functions (continued)

Function	AIX	HP	Linux (x386 & S/390)	OS/2	OS/390	OS/400	SUN	Win NT
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_COPY	X	X	X	X	X	X	X	X
DTWF_DELETE	X	X	X	X	X	X	X	X
DTWF_EXISTS	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_READFILE	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
DTWF_WRITEFILE	X	X	X	X	X	X	X	X
DTWR_ADDENTRY	X			X		X		X
DTWR_CLEARREG	X			X		X		X
DTWR_CLOSEREG						X		
DTWR_CREATEREG	X			X		X		X
DTWR_DELENTRY	X			X		X		X
DTWR_DELREG	X			X		X		X
DTWR_LISTREG	X			X		X		X
DTWR_OPENREG						X		
DTWR_RTVENTRY	X			X		X		X
DTWR_UPDATEENTRY	X			X		X		X

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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
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 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX
AS/400
DB2
DB2 Universal Database
DRDA
DataJoiner
IBM
IMS
Language Environment
MVS/ESA
Net.Data
OS/2
OS/390
OS/400
OpenEdition

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

absolute path
The full path name of an object. Absolute path names begins at the highest level, or "root" directory (which is identified by the forward slash (/) or back slash (\) character).

ANSI American National Standard for Information Systems

API Application Programming Interface

applet A Java program included in an HTML page. Applets work with Java-enabled browsers, such as Netscape Navigator, and are loaded when the HTML page is processed.

BLOB Binary large object.

CGI Common Gateway Interface.

CLOB Character large object.

commitment control
The establishment of a boundary within the process that Net.Data is running under where operations on resources are part of a unit of work.

Common Gateway Interface (CGI)
A standardized way for a Web server to pass control to an application program and receive data back.

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*

current working directory
The default directory of a process from which all relative path names are resolved.

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.

DATALINK
A DB2 data type that enables logical references from the database to a file stored outside the database.

data type
An attribute of columns and literals.

DBCLOB
Double-byte character large object.

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.

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.

Internet
An international public TCP/IP computer network.

Intranet
A TCP/IP network inside a company firewall.

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 REXX.

LOB Large object.

middleware
Software that mediates between an application program and a network. It manages the interaction between a client application program and a server through the network.

null A special value that indicates the absence of information.

path A search route used to locate files.

path name
Tells the system how to locate an object. The path name is expressed as a sequence of directory names followed by the name of the object. Individual directories and the object name are separated by a forward slash (/) or back slash (\) character.

persistence
The state of keeping an assigned value for an entire transaction, where a transaction spans multiple Net.Data invocations. Only variables can be persistent. In addition, operations on resources affected by commitment control are kept active until an explicit commit or rollback is done, or when the transaction completes.

port A 16-bit number used to communicate between TCP/IP and a higher level protocol or application.

registry
A repository where strings can be stored and retrieved.

relative path name
A path name that does not begin at the highest level, or "root" directory. The system assumes that the path name begins at the process's current working directory.

secure endpoint URL
Endpoint beginning with https

SSL Secure Sockets Layer

TCPIP Transmission Control Protocol/Internet Protocol

uniform resource locator
An address that names a HTTP server and optionally a directory and file name, for example: `http://www.ibm.com/software/data/net.data/index.html`.

unit of work
A recoverable sequence of operations that are treated as one atomic operation. All operations within the unit of work can be completed (commited) or undone (rolled back) as if the operations are a single operation. Only operations on resources that are affected by commitment control can be committed or rolled back.

URL Uniform resource locator.

Web server
A computer running HTTP server software, such as Internet Connection.

wire All the underlying components that are responsible for physically sending or receiving a message on the web

XML eXtensible Mark-up Language

Index

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Printed in USA