Net.Data Administration and Programming Guide for OS/2, Windows NT, and UNIX

Net.Data Administration and Programming Guide for OS/2, Windows NT, and UNIX

Note

Be sure to read the information in “Appendix E. Notices” on page 153 before using this information and the product it supports.

Fourth Edition (June 1998)

Contents

Preface

Thank you for selecting Net.Data Version 2, IBM's development 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.

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

About Net.Data

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

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

A request, in the form of a URL (uniform resource locator), flows from a browser, such as Netscape 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 Perl scripts, C and C++ applications, or REXX programs
- Access databases such as DB2

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 supports industry-standard interfaces such as HyperText Transfer Protocol (HTTP) and Common Gateway Interface (CGI). HTTP is used between the browser and the Web server, and CGI is used between the Web server and Net.Data. This support lets you select your favorite browser or Web server for use with Net.Data. The Net.Data family of products provide similar capablities on the OS/400, OS/390, Windows NT, AIX, OS/2, HP-UX, Sun Solaris, and SCO operating systems. Net.Data also supports FastCGI and the major Web server Application Programming Interfaces (APIs) on multiple operating systems.

In addition, Net.Data Version 2 includes other performance enhancements to meet your application's requirements, including:

- Caching of HTML pages
- Invocation of Net.Data without a macro file (direct request)
- Minimization of extraneous white space within generated Web pages
- Optimized math functions

Net.Data Version 2 also includes a number of functional enhancements:

- Language environments enhancements include the ability to retrieve one or more result sets from stored procedures through the SQL or ODBC language environments.
- Macro language enhancements include:
  - Ability to place comments anywhere
  - Nested IF blocks
  - WHILE blocks
  - Ability to receive a single or multiple result set from a stored procedure
  - DBCS-enabled string and word functions
  - Support for the SQL decimal datatype in parameter lists for stored procedures
  - Support for cookies
  - Support for dynamically generated e-mail messages

A graphical administration tool helps you administer Net.Data configuration settings for the AIX, Windows NT, and OS/2 operating systems. The administration tool also assists you in specifying security for your connections to databases that use Live Connection.

To help you easily access data from your database, Net.Data provides a variety of tools, including NetObjects Fusion plug-ins and smartguides for Java-based development. These tools work with the Net.Data Java servlets in the Java environment, allowing you to create applications that are portable across operating systems. NetObjects Fusion plug-ins allow you to use the NetObjects Fusion Web development tool to build sophisticated applications with dynamic data from relational data sources. Net.Data smartguides provide a graphical tool to guide you through creating basic Net.Data macros.

About This Book

This book discusses administration and programming concepts for Net.Data, as well as how to configure Net.Data and its components, plan for security, and improve performance.

Building on your knowledge of programming languages and database, you learn how to use the Net.Data macro language or Java servlets to develop macros. You learn how to use Net.Data-provided language environments that access DB2 databases and IMS transactions using IMS Web, as well as use Java, REXX, Perl, and other programming languages to access your data.

This book may 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.software.ibm.com/data/net.data

Who Should Read This Book

This book is intended for people involved in planning and writing Net.Data applications. Operating system differences, Net.Data messages, and other information are described in *Net.Data Reference*.

To understand the concepts discussed in this book, you should be familiar with how a Web server works, understand simple SQL statements, and know HTML tags, including HTML form tags. Additionally, you should become familiar with the information in *Net.Data Reference* and *Net.Data Language Environment Reference*

About Examples in This Book

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

Chapter 1. What is Net.Data?

With HTML alone, you can create static Web pages; in other words, pages that do not change unless you edit them. To put “live” data and applications on the Web (such as current sales statistics), Web site developers usually write programs that execute as middleware at the Web server to dynamically build Web pages. Writing these types of programs is not easy.

Net.Data simplifies the writing of interactive Web applications through *macros*. With Net.Data macros, you can use logic, variables, function calls, and report-generating tools. A macro is a text file containing Net.Data macro language constructs, HTML tags, and language environment statements, such as SQL and Perl. Net.Data processes the macro file to produce HTML output. Macros combine the simplicity of HTML with the dynamic functionality of Web server programs, making it easy to add live data to static Web pages. The live data can be extracted from local or remote databases and from flat files, or be generated by applications and system services.

Figure 1 illustrates the relationship between Net.Data and the Web server, and to the supported data and programming language environments.

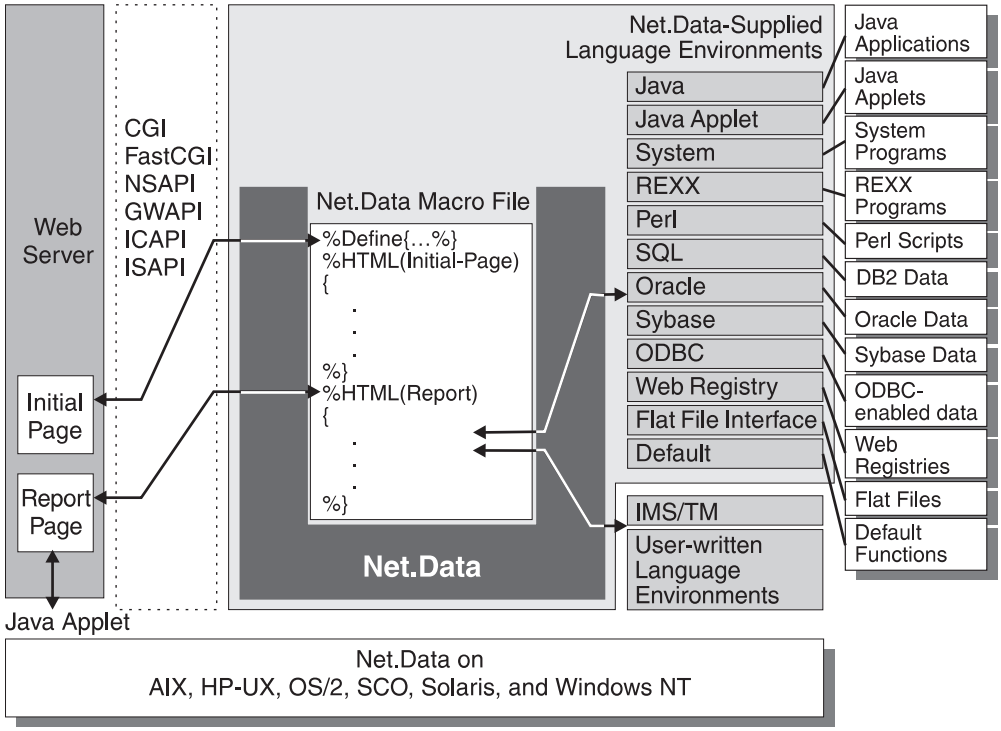


Figure 1. The Relationship between Net.Data, the Web Server, and Supported Data and Program Sources

The Web server invokes Net.Data as a CGI or FastCGI process, or a Web server application programming interface (API) thread by calling Net.Data as a DLL or shared library when it receives a URL that requests Net.Data services. The URL includes Net.Data-specific information, including either the macro file that is to be processed or the SQL statement or program that is to be directly invoked. When Net.Data finishes processing the request, it sends the resulting Web page to the Web server. The server passes it on to the Web client, where it is displayed by the browser.

Net.Data is a good choice for creating dynamic Web pages because using the macro language is simpler than writing your own Web server applications and because Net.Data lets you use languages that you already know, such as HTML, SQL, Perl, REXX, and JavaScript. Net.Data also provides language environments that access DB2 databases, execute IMS transactions using IMS Web, or use REXX, Perl, and other languages for your applications.

Another important advantage is that Net.Data supports access to many different database sources, allowing Web developers to work with data from a variety of databases including DB2, IMS, Oracle, Sybase, and any ODBC-enabled data source. See *Net.Data Language Environment Reference* for more information about Net.Data-provided language environments.

The Net.Data Web application environment provides the following features:

Interpreted Macro Language

The Net.Data macro language is an interpreted language. When Net.Data is invoked to process a macro, Net.Data directly interprets each language statement in a sequential fashion, starting from the top of the file. Using this approach, any changes you make to a macro can be immediately seen when you next specify the URL that executes the macro. No recompilation is required.

Direct Requests

Simple requests that require the execution of a single SQL statement, DB2 stored procedure, REXX program, C or C++ program, or Perl script do not require the creation of a macro. These requests can be specified directly within the URL that flows from the browser to the Web server.

Free Format

The Net.Data macro language has only a few rules about programming format. This simplicity provides programmers with freedom and flexibility. A single instruction can span many lines, or multiple instructions can be entered on a single line. Instructions can begin in any column. Spaces or entire lines can be skipped. Comments can be used anywhere.

Variables Without Type

Net.Data regards all data as character strings. Net.Data uses built-in functions to perform arithmetic operations on a string that represents a valid number, including those in exponential formats. Macro language variables are discussed in detail in “Net.Data Macro Variables” on page 62.

Built-in Functions

Net.Data supplies built-in functions that perform various processing, searching, and comparison operations for both text and numbers. Other built-in functions provide formatting capabilities and arithmetic calculations.

Error Handling

When Net.Data detects an error, messages with explanations are returned to the client. You can customize the error messages before they are returned to a user at a browser. See *Net.Data Reference* for more information.

Chapter 2. Configuring Net.Data

You can install Net.Data for your operating system by using the instructions in the README file that accompanied the product. Most configuration steps are completed during installation; this varies by operating system.

After installing Net.Data for your operating system, you must configure Net.Data and modify your configuration for the Web server. Configuration tasks include:

- Customizing the Net.Data initialization (INI) file
- Configuring Net.Data for use with FastCGI or one of the support Web server APIs (optional)
- Customizing the Web server configuration and environment variable files
- Configuring the Cache Manager (optional)
- Configuring Live Connection (optional)
- Specifying access rights

You use the following tools to configure Net.Data:

- A text editor

  Use a text editor to edit the INI file and the Live Connection configuration file on all operating systems. You also use a text editor to update any Web server configuration files. It is a good idea to back up the files before you make changes.
- The Net.Data administration tool

  The administration tool provides a graphical interface for customizing the INI file and the Live Connection configuration file. You can use the administration tool to configure Net.Data on the OS/2, Windows NT, and AIX operating systems.

The method you use depends on which components need to be configured and the operating system Net.Data is running on, as described in Table 1. If you start using one particular method for a configuration task, you should continue to use that method for the best results.

Table 1. Comparison of configuration methods with tasks and operating systems. **A** *- Can be configured with the administration tool or manually.* **M** *- Can be configured manually, only.*

Task	**Platforms:**			
	AIX	**NT**	**OS/2**	**HP|SUN|SCO**
Configure the Net.Data INI file	A	A	A	M
Define cliettes ports	A	A	A	M
Define cliettes	A	A	A	M
Turn on cliette password encryption	A	N/A	N/A	N/A
Turn on error logging	A	A	A	M
Configure Web Server for FastCGI, CGI, and APIs*	M	M	M	M
Define Cache Manager Ports	M	M	N/A	N/A
Configure Cache Manager	M	M	N/A	N/A

*__Tip:__ Many Web servers have administration tools that you can use to configure the Web server.

This chapter describes how to configure Net.Data and how to modify your configuration of the Web server for use with Net.Data. Additionally, it describes how to configure optional components.

About the Net.Data Initialization File

Net.Data uses its initialization file to establish the settings of various configuration variables and to configure language environments and search paths. The settings of configuration variables control various aspects of Net.Data operation, such as the encoding of character data within Web pages, whether string and word functions are DBCS enabled, and the name of the DB2 instance for access to database data. The settings also specify how Net.Data connects and communicates with the language environments, databases, connection management, and caching, and specifies whether error logging is activated.

The language environment statements define the Net.Data language environments that are available and identify special input and output parameter values that flow to and from the language environments. The path statements specify the directory paths to files that Net.Data uses, such as macros, REXX programs, and Perl scripts.

The Net.Data initialization file, db2www.ini, is located in the Web server's document directory. See the README file for you operating system for more information.

Authorization Tip: Ensure that the Web server has access rights to this file. See "Specifying Access Rights to Net.Data Files" on page 40 for more information.

About the Net.Data Configuration Files for Optional Components

The following sections discuss the configuration files for optional components of Net.Data.

The Live Connection Configuration File

Live Connection provides connection management on Windows NT, OS/2, and UNIX operating systems to improve performance by eliminating start-up overhead. The Net.Data Live Connection configuration file contains information about one or more named cliettes. A cliette is a long-running process that maintains a connection to a database or a Java application that endures over Net.Data macro invocations

from multiple users. After a cliette is started, it continues to exist until Net.Data Live Connection terminates. Multiple cliettes can connect to a single database.

As part of the cliette information in the configuration file, you specify a cliette name, unique ports, and the minimum and maximum number of processes. For database cliettes, you can also specify the database name, login, and password for each cliette entry. On AIX, you can encrypt the password.

Authorization Tip: Ensure that the user ID that starts Connection Manager has access rights to this file. See "Specifying Access Rights to Net.Data Files" on page 40 for more information.

The Cache Manager Configuration File

The Cache Manager configuration file contains the definitions for the Cache Manager and each of the caches. Net.Data caching is described in "Chapter 9. Net.Data Caching" on page 107. Configuring the Cache Manager is described in "Configuring the Cache Manager and Net.Data Caches" on page 112. The structure of the file is a series of sections, or stanzas:

Cache Manager stanza
: This stanza defines the parameters of the Cache Manager itself and includes network information, logging status, and tracing status. The stanza is required and must be labeled `cache-manager`.

Cache definition stanzas
: These stanzas define the parameters for each cache; one cache definition stanza in the configuration file exists for each cache that is managed by the Cache Manager; this section contains network information, memory and space requirements, logging status, and statistics status. The cache definition stanza is required for each cache that is managed by the Cache Manager.

The Cache Manager configuration file is not managed by the administration tool and can be updated with any text editor. See "Chapter 9. Net.Data Caching" on page 107 to learn how to define this file.

Authorization Tip: Ensure that the user ID that starts the Cache Manager has access rights to this file. See "Specifying Access Rights to Net.Data Files" on page 40 for more information.

Common Sections of the Net.Data Initialization, Control, and Macro Files

Certain portions of the Net.Data initialization, configuration and macro files must be consistent for all components of Net.Data to work as a whole. The following table summarizes the areas of each of these files that must match.

Table 2. Consistency Requirements for the Net.Data Configuration Files and the Macro File

File	Common Sections	Notes
Net.Data INI File	Environment Statement	The language environments that use Live Connection must specify the database cliette name in their environment statement
	Live Connection configuration variables	When using Net.Data Live Connection, specify the Live Connection port, DTW_CM_PORT. This variable value must match the MAIN_PORT value in the Live Connection configuration file.
	Cache configuration variables	When using Net.Data caching, optionally include port and machine variables. These values must match those used in the Cache Manager configuration file, if used.
Live Connection Configuration File	Cliette Definitions	Each cliette definition must match a corresponding definition in the INI file. Additionally, the MAIN_PORT value must match the DTW_CM_PORT variable value in the INI file.
Cache Manager Configuration File	Cache Manager Configuration Variables	When using Net.Data caching, you can optionally include port and machine variables. These values must match those used in the INI file, if used.

The following fragments illustrate the relationship between a macro file, an initialization file, and a Live Connection configuration file. Two cliettes are used by the macro file (`DTW_SQL:SAMPLE, DTW_SQL:CELDIAL`) and access two DB2 databases, called `SAMPLE` and `CELDIAL`. The Live Connection configuration file contains the cliette names and definitions. The ENVIRONMENT statement in the Net.Data initialization file refers to the cliette name. The LOGIN and PASSWORD values are specified in the Live Connection configuration file.

Figure 2 on page 7 shows a fragment of the macro file that contains the @DTW_ASSIGN statement that defines which cliette is to be used to access a database.

```
<|*********************************************>
<|** This is an HTML comment                    **>
<|** Access the SAMPLE database using       **>
<|** cliette DTW_SQL:SAMPLE                 **>
<|*********************************************>
@DTW_ASSIGN (DATABASE, ″ SAMPLE ″)
@insert_customer
(customer_name, customer_street, customer_city, customer_state,
customer_country, customer_zip, customer_credit, customer_expiry)


<|*********************************************>
<|** This is an HTML comment                    **>
<|** Process the CELDIAL database using      **>
<|** the cliette DTW_SQL:CELDIAL **>
<|*********************************************>
@DTW_ASSIGN (DATABASE, ″ CELDIAL ″)
@insert_customer
(customer_name, customer_street, customer_city, customer_state,
customer_country, customer_zip, customer_credit, customer_expiry)
```

Figure 2. Net.Data Macro File Fragment

Note that the DATABASE configuration variable will substituted into the ENVIRONMENT statement of the INI file to generate the cliette name. This allows you to access multiple databases from the same macro file.

Figure 3 shows a fragment of the Net.Data initialization file that contains the ENVIRONMENT statement and the associated cliette type. There is one ENVIRONMENT statement for each cliette type in the initialization file. For each database cliette type, the ENVIRONMENT statement specifies a cliette name. The name is made up of the cliette type and a variable reference, $(DATABASE), which is resolved at run time. Each language environment that uses Live Connection must have a cliette definition in the ENVIRONMENT statement.

```
ENVIRONMENT (DTW_SQL)
    (IN DATABASE, LOGIN, PASSWORD, TRANSACTION_SCOPE, SHOWSQL,
    ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
    CLIETTE ″DTW_SQL:$(DATABASE)″
```

Figure 3. Net.Data Initialization File Fragment

Figure 4 on page 8 shows a fragment of the Live Connection configuration file, which contains the cliette definitions for DTW_SQL:SAMPLE and DTW_SQL:CELDIAL.

```
CONNECTION_MANAGER{
        MAIN_PORT=7128
        ADMIN_PORT1=7131
        ADMIN_PORT2=7133
        ENCRYPTION=OFF
        }

        ######################################################################
        # This is a comment in a Live Connection configuration file.
        # Comments start with a pound (hash) character.
        # Comments terminate at the end of the line and do not continue to
        # the next line unless another pound (hash) character is specified.
        # You can include comments at the end of lines containing Live
        # Connection keywords except on password lines.
        # You cannot include comments anywhere on lines containing the
        # password keyword.
        # You cannot include spaces and pound (hash) characters within any
        # name, such as cliette name or in database cliette passwords.
       ######################################################################
        CLIETTE  DTW_SQL:SAMPLE{
        MIN_PROCESS=1
        MAX_PROCESS=3
        START_PRIVATE_PORT=7100
        START_PUBLIC_PORT=7300
        EXEC_NAME=dtwcdb2.exe
        DATABASE=SAMPLE
                LOGIN=USER1
        PASSWORD=HAMLET
        }

        CLIETTE  DTW_SQL:CELDIAL{
        MIN_PROCESS=1
        MAX_PROCESS=5
        START_PRIVATE_PORT=7500
        START_PUBLIC_PORT=7700
        EXEC_NAME=dtwcdb2.exe
        DATABASE=CELDIAL
        LOGIN=USER2
        PASSWORD=OPHELIA
        }
```

Figure 4. Live Connection configuration file fragment

Customizing the Net.Data Initialization File

The information contained in the initialization file is specified using three types of configuration statements, described in the following sections:

- "Configuration Variable Statements" on page 9
- "Customizing Path Configuration Statements" on page 13
- "Environment Configuration Statements" on page 16

The sample initialization file shown in Figure 5 on page 9 contains examples of these statements and is valid for for OS/2 and Windows NT.

```
1  DTW_CM_PORT 7128
2  DTW_INST_DIR c:\db2www
3  DTW_LOG_DIR c:\db2www\logs
4  DB2INSTANCE DB2
5  MACRO_PATH c:\DB2WWW\Macro
6  HTML_PATH  c:\www\html
7  INCLUDE_PATH c:\db2www\Macro
8  EXEC_PATH c:\db2www\Macro
9  FFI_PATH c:\pub\ffi;pub\ffi\data
10 ENVIRONMENT (DTW_SQL)          [DLL path]   [Parameter list]
11 ENVIRONMENT (DTW_SYB)          [DLL path]   [Parameter list]
12 ENVIRONMENT (DTW_ORA)          [DLL path]   [Parameter list]
13 ENVIRONMENT (DTW_ODBC)         [DLL path]   [Parameter list]
14 ENVIRONMENT (DTW_DEFAULT)      [DLL path]   [Parameter list]
15 ENVIRONMENT (DTW_APPLET)       [DLL path]   [Parameter list]
16 ENVIRONMENT (DTW_REXX)         [DLL path]   [Parameter list]
17 ENVIRONMENT (DTW_PERL)         [DLL path]   [Parameter list]
18 ENVIRONMENT (DTW_SYSTEM)       [DLL path]   [Parameter list]
19 ENVIRONMENT (DTW_FILE)         [DLL path]   [Parameter list]
20 ENVIRONMENT (DTW_WEBREG)       [DLL path]   [Parameter list]
21 ENVIRONMENT (DTW_JAVAPPS)      [Parameter list]
22 ENVIRONMENT (DTW_DLDPB)        [Parameter list]
23 ENVIRONMENT (USR_TEST)         [Parameter list]
```

- Lines 1 - 4 define configuration variables
- Lines 5 - 9 define paths to files required to process the macro
- Lines 10 - 23 define the environment statements that are available.

Figure 5. The Net.Data initialization file

The following sections describe how to customize the configuration statements in the INI file.

Configuration Variable Statements

Net.Data configuration variable statements set the values of configuration variables. Configuration variables are used for various purposes. Some variables are required by a language environment to work properly or to operate in an alternate mode. Other variables control the character encoding or content of the Web page being constructed. Additionally, you can use configuration variable statements to define application-specific variables.

The configuration variables you use depend on the language environments, and databases you are using, as well as other factors that are specific to the application.

To update the configuration variable statements: Customize the initialization file with the configuration variables that are required for your application. A configuration variable has the following syntax:

NAME[=]*value-string*

The equal sign is optional, as denoted by the brackets.

The following sub-sections describe the configuration variables statements that you can use in the initialization file:

Cache Manager Configuration Variables

Two optional configuration variables are used if the Cache Manager runs on a machine other than where the Net.Data macro runs:

- CACHE_PORT specifies which port number Net.Data uses to connect to the Cache Manager.
- CACHE_MACHINE specifies the TCP/IP host name of the local or remote machine.

Table 3 describes the options for specifying machine IDs and port numbers for these variables.

Table 3. Cache Manager Configuration Variables: Configuration Options

Default Connection Manager Values	If the cache machine is specified ...	If the cache machine is not specified ...
If the cache port is specified ...	Net.Data connects to the Cache Manager on the specified machine using the specified port.	Net.Data connects to the Cache Manager on the local machine using the specified port.
If the cache port is not specified ...	Net.Data connects to the Cache Manager on the specified machine using the default port of 7175.	Net.Data connects to the Cache Manager on the local machine using the default port of 7175.

If the Cache Manager runs on the local machine, UNIX-domain sockets or named pipes are used for communication and no configuration is necessary.

The Cache Manager runs on AIX and Windows NT machines, only. See "Chapter 9. Net.Data Caching" on page 107 to learn about Net.Data caching.

CACHE_PORT: Cache Manager Port Variable

Specifies the TCP/IP port the Cache Manager is listening on. This port number must match the port number specified in the Cache Manager configuration file, so Net.Data can communicate with the Cache Manager. If not specified, Cache Manager uses the default port 7175.

Syntax:

```
CACHE_PORT port_number
```

Parameter:

port_number

A unique port number assigned to the Cache Manager to service cache requests. The default value is 7175.

CACHE_MACHINE: Cache Manager Machine ID Variable

Specifies the machine where the Cache Manager resides. If not specified, Net.Data assumes the correct machine is the local machine.

Syntax:

```
CACHE_MACHINE host_name
```

Parameter:

host_name
The qualified TCP/IP host name of the local or remote machine where the Cache Manager runs. The default value is the local machine's host name.

DB2INSTANCE: DB2 Instance Variable

Specifies the instance of DB2 used by the SQL language environment. This variable value is required when Net.Data connects to DB2 running on the Windows NT, OS/2, and UNIX operating systems.

DB2 on the OS/2, Windows NT, and UNIX operating systems needs DB2INSTANCE to be defined as an environment variable. If Net.Data detects that DB2INSTANCE is not defined as an environment variable, it will set the DB2INSTANCE environment variable to the value of DB2INSTANCE found in the INI file before attempting to connect to DB2.

Syntax:

```
DB2INSTANCE instance_name
```

DTW_CM_PORT: Live Connection Port Number Variable

Specifies a unique port number that Net.Data uses for Live Connection.

Syntax:

```
DTW_CM_PORT port_number
```

Where *port_number* specifies the unique port number used for Live Connection.

DTW_INST_DIR: Net.Data Installation Directory Variable

The DTW_INST_DIR variable in the Net.Data INI file is used to locate certain files during Net.Data execution. You set this variable at installation time to specify the home directory, *<inst_dir>*, where Net.Data is installed. Do not change this value after installation.

DTW_LOG_DIR: Error Log Location Variable

Specify the directory where the error logs are stored with the DTW_LOG_DIR configuration variable. When logging is enabled with the DTW_LOG_LEVEL variable in the macro file, the log files are stored in the directory specified in the path statement of the DTW_LOG_DIR variable. The default is `\inst_dir\logs\netdata.logs`. See "Chapter 11. Logging Net.Data Error Messages" on page 135 to learn about logging error messages with Net.Data and about the DTW_LOG_LEVEL variable.

Requirement: The DTW_LOG_DIR variable must be defined for Net.Data to log files. If not defined, logging does not occur even if DTW_LOG_LEVEL is set to ERROR or WARNING in the macro file.

Syntax:

```
DTW_LOG_DIR \inst_dir\path
```

Example: Initialization file configuration

```
DTW_LOG_DIR \inst_dir\mylogfiles\
```

DTW_MBMODE: Native Language Support Variable

Activates national language support for word and string functions. When the value of this variable is YES, all string and word functions correctly process DBCS characters within strings by treating strings as mixed data (that is, as strings that potentially contain characters from both single-byte character sets and double byte character sets). The default value is NO. You can override the value set in the initialization file by setting the DTW_MBMODE variable in a Net.Data macro file.

Syntax:

```
DTW_MBMODE [=] NO|YES
```

Example: Activating national language support

```
DTW_MBMODE = YES
```

To override the initialization file setting in a macro file:

- Add the DTW_MBMODE variable as an IN parameter for the DEFAULT ENVIRONMENT statement in initialization file as shown in the following example:

  ```
  ENVIRONMENT (DTW_DEFAULT) DTWFUNC.DLL
     (IN DTW_MBMODE, OUT RETURN_CODE )
  ```

- In a macro file, set the configuration variable DTW_MBMODE to the value required for the application.

DTW_OPTIMIZE_MATH: Optimize Math Functions Variable

Optimizes the performance of mathematical functions. When DTW_OPTIMIZE_MATH is set to YES, Net.Data uses C mathematical formatting and the functions run faster; however the output format is different than without this variable.

- In particular, trailing zeros after decimal points are not displayed.
- Precision always means the number of significant digits.

When DTW_OPTIMIZE_MATH is set to NO, Net.Data uses REXX mathematical formatting. Functions run slower, but provide output formats that are consistent with the output generated by previous versions of Net.Data. The default value is NO.

Syntax:

```
DTW_OPTIMIZE_MATH NO|YES
```

To override the initialization file setting in a macro file:

In a macro file, set the configuration variable DTW_OPTIMIZE_MATH, using DTW_ASSIGN, to the value required for the application.

DTW_SMTP_SERVER: E-mail SMTP Server Variable

Specifies the SMTP server to use for sending out e-mail messages. The value of this variable can either be a hostname or an IP address. If this variable is not set, Net.Data uses the local host as the SMTP server.

Syntax:

```
DTW_SMTP_SERVER server_name
```

Where *server_name* is the hostname or IP address of the SMTP server to be used for sending e-mail messages.

Example:

```
DTW_SMTP_SERVER = "myserver"
```

Customizing Path Configuration Statements

Net.Data determines the location of files and executable programs used by Net.Data macro files from the settings of path configuration statements. The path statements are:

- MACRO_PATH
- EXEC_PATH
- INCLUDE_PATH
- FFI_PATH
- HTML_PATH

These path statements identify one or more directories that Net.Data searches when attempting to locate macro files, executable files, text files, LOB files, and include files. The path statements that you need depend on the Net.Data capabilities that your macros use.

Update guidelines:

Several general guidelines apply to all path statements.

- Each specified directory is delimited by a semicolon (;).
- Forward slashes (/) and back slashes (\) are treated the same.
- Each path statement can specify multiple paths, except for the HTML_PATH, which can have only one path statement. Paths are searched from left to right in the order specified. This multiple-path capability lets you organize your files within multiple directories. For example, you can place each of your Web applications in its own directory.
- It is recommended to use absolute path statements.

Tip: Net.Data searches all specified directories, but not the subdirectories. For example, if you have Net.Data macros in the following directories, you must specify each subdirectory in the path statement:

```
/usr/test/client
/usr/test/assoc
/usr/test/partner
```

Your MACRO_PATH statement might look like this:

```
MACRO_PATH [=] /usr/test/client;usr/test/assoc;usr/test/partner
```

The following sections describe the purpose and syntax of each path statement and provide examples of valid path statements. The examples can differ from your application, depending on your operating system and configuration.

MACRO_PATH

The MACRO_PATH configuration statement identifies the directories that Net.Data searches for Net.Data macro files. For example, specifying the following URL requests the Net.Data macro with the path and file name `macro/sqlm.d2w`:

```
http://server/cgi-bin/db2www/macro/sqlm.d2w/report
```

Syntax:

```
MACRO_PATH [=] path1;path2;...;pathn
```

The equal sign (=) is optional, as indicated by brackets.

Net.Data appends the path `macro/sqlm.d2w` to the paths in the MACRO_PATH configuration statement, from left to right until Net.Data finds the macro file or searches all paths. See “Chapter 4. Invoking Net.Data” on page 47 for information on invoking Net.Data macros.

Example: The following example shows the MACRO PATH statement in the initialization file and the related link that invokes Net.Data.

Net.Data initialization file:

```
MACRO_PATH = /u/user1/macros;/usr/lpp/netdata/macros;
```

HTML link:

```
<A HREF="http://server/cgi-bin/db2www/query.d2w/input">Submit another query.</A>
```

If the file *query.d2w* is found in the the directory `/u/SYSADM/macros`, then the fully-qualified path is `/u/SYSADM/macros/query.d2w`.

EXEC_PATH

The EXEC_PATH configuration statement identifies one or more directories that Net.Data searches for an external program that is invoked by the EXEC statement or an executable variable. The order of the directories in the path statement determines the order Net.Data searches for the directories. If the program is found, the external program name is appended to the path specification, resulting in a fully qualified file name that is passed to the language environment for execution.

Syntax:

```
EXEC_PATH [=] path1;path2;...;pathn
```

Example: The following example shows the EXEC PATH statement in the initialization file and the EXEC statement in the macro file that invokes an external program.

Net.Data initialization file:

```
EXEC_PATH = /u/user1/prgms;/usr/lpp/netdata/prgms;
```

Net.Data macro:

```
%FUNCTION(DTW_REXX) myFunction() {
  %EXEC{ myFunction.cmd %}
%}
```

If the file *myFunction.cmd* is found in the `/usr/lpp/netdata/prgms` directory, the qualified name of the program is `/usr/lpp/netdata/prgms/myFunction.cmd`.

INCLUDE_PATH

The INCLUDE_PATH configuration statement identifies one or more directories that Net.Data searches, in the order in which they are specified, to find a file specified

on an INCLUDE statement in a Net.Data macro. When it finds the file, Net.Data appends the include file name to the path specification to produce the qualified include file name.

Syntax:

```
INCLUDE_PATH [=] path1;path2;...;pathn
```

Tip: If you are including HTML files from a local Web server, use the INCLUDE_URL construct as shown in the local Web server example for INCLUDE_URL in *Net.Data Reference*. By using the demonstrated syntax, you do not need to update the INCLUDE_PATH to specify directories that are already known to the Web server.

Example 1: The following example shows both the INCLUDE_PATH statement in the initialization file and the INCLUDE statement that specifies the include file.

Net.Data initialization file:

```
INCLUDE_PATH = /u/SYSADM/includes;/usr/lpp/netdata/includes;
```

Net.Data macro:

```
%INCLUDE "myInclude.txt"
```

If the file *myInclude.txt* is found in the `/u/SYSADM/includes` directory, the fully-qualified name of the include file is `/u/SYSADM/includes/myInclude.txt`.

Example 2: The following example shows the INCLUDE_PATH statement and an INCLUDE file fully-qualified by a subdirectory name.

Net.Data initialization file:

```
INCLUDE_PATH = /u/SYSADM/includes;/usr/lpp/netdata/includes;
```

Net.Data macro:

```
%INCLUDE "/OE/oeheader.inc"
```

The include file is searched for in the directories `/u/SYSADM/includes/OE` and `/usr/lpp/netdata/includes/OE`. If the file is found in `/usr/lpp/netdata/includes/OE`, the fully qualified name of the include file is `/usr/lpp/netdata/includes/OE/oeheader.inc`.

FFI_PATH

The FFI_PATH configuration statement identifies one or more directories that Net.Data searches, in the order in which they are specified, for a flat file that is referenced by a flat file interface (FFI) function.

Syntax:

```
FFI_PATH [=] path1;path2;...;pathn
```

Example: The following example shows an FFI PATH statement in the initialization file.

Net.Data initialization file:

```
FFI_PATH = /u/SYSADM/ffi;/usr/lpp/netdata/ffi;
```

When the FFI language environment is called, Net.Data looks in the path specified in the FFI_PATH statement.

HTML_PATH

The HTML_PATH configuration statement specifies into which directory Net.Data writes large objects (LOBs). This value can be changed by configuring the initialization file. This path statement accepts only one directory path.

Syntax:

```
HTML_PATH [=] path
```

Example: The following example shows the HTML PATH statement in the initialization file.

Net.Data initialization file:

```
HTML_PATH = /pub/htm
```

When a query returns a LOB, Net.Data saves it in the directory specified in the HTML_PATH configuration statement.

Performance tip: Consider system limitations when using LOBs because they can quickly consume resources. See “Using Large Objects” on page 91 for more information.

Environment Configuration Statements

An ENVIRONMENT statement configures a language environment. A language environment is a component of Net.Data that Net.Data uses to access a data source such as a DB2 database or to execute a program written in a language such as REXX. Net.Data provides a set of language environments, as well as an interface that allows you to create your own language environments. These language environments and the language environment interface are described in *Net.Data Language Environment Reference*.

Net.Data requires that an ENVIRONMENT statement exist in the Net.Data initialization file for a language environment before you can invoke the language environment.

Net.Data specifies several variables that affect the way in which Net.Data language environments interpret calls to functions that are defined in FUNCTION blocks. The settings of these variables must be passed to a language environment to have an effect.

For example, a macro can define a DATABASE variable to specify the name of a database at which an SQL statement within a DTW_SQL function is to be executed. The value of DATABASE must be passed to the SQL language environment (DTW_SQL) so that the SQL language environment can connect to the designated database. To pass the variable to the language environment, you must add the DATABASE variable to the parameter list of the environment statement for DTW_SQL.

The sample Net.Data initialization file makes several assumptions about customizing the setting of Net.Data environment configuration statements. These assumptions may not be correct for your environment. Modify the statements appropriately for your environment.

To add or update an ENVIRONMENT statement:

ENVIRONMENT statements have the following syntax:

```
ENVIRONMENT(type) library_name (parameter_list, ...) [CLIETTE "cliette_name"]
```

Parameters:

- *type*

  The name by which Net.Data associates this language environment with a FUNCTION block that is defined in a Net.Data macro. You must specify the type of the language environment on a FUNCTION block definition to identify the language environment that Net.Data should use to execute the function.

- *library_name*

  The name of the DLL or shared library containing the language environment interfaces that Net.Data calls. In OS/2, the DLL name is specified without the *.dll* extension. In AIX, the name of the shared object is specified with the *.o* extension,

- *parameter_list*

  The list of parameters that are passed to the language environment on each function call, in addition to the parameters that are specified in the FUNCTION block definition.

  The parameters are passed in the *parm_data_array* field of the *dtw_lei* structure, following the parameters that are specified in the FUNCTION block definition. (See *Net.Data Language Environment Reference* for information about these structures.)

  You must define these parameters as configuration variables or as variables in your macro file before executing a function that will be processed by the language environment. If a function modifies any of its output parameters, the parameters keep their modified value after the function completes.

- *cliette_name*

  The name of the cliette. The *cliette_name* can refer to the Java Application language environment cliette, or it can be a database cliette. The *cliette_name* parameter is used with the CLIETTE keyword, both of which are only used with Live Connection. `CLIETTE` and *`cliette_name`* are optional and can be specified only for database and Java application language environments, and for user-defined language environments that have cliettes written for them.

  Java Application cliette

  This cliette name specifies the Java Application language environment.

  Syntax:

  ```
  CLIETTE "DTW_JAVAPPS"
  ```

  Database cliette

  This cliette name specifies a cliette that is associated with a database.

  Syntax:

  ```
  CLIETTE "type:db_name"
  ```

  Parameters:

type The database language environment associated with the cliette. See 39 for a list of valid types.

db_name
The database cliette name. This name is often the same as the database with which the cliette is associated, such as MYDBASE, but can also be another name. *db_name* is optional when using the Oracle language environment.

When Net.Data processes the INI file, it does not load the language environment DLLs or shared libraries. Net.Data loads a language environment DLL when it first executes a function that identifies that language environment. The DLL then remains loaded for as long as Net.Data is loaded.

Example: ENVIRONMENT statements for Net.Data-provided language environments

When customizing the ENVIRONMENT statements for your application, add the variables on the ENVIRONMENT statements that need to be passed from your initialization file to a language environment or that Net.Data macro writers need to set or override in their macros.

```
ENVIRONMENT (DTW_SQL)      DTWSQL    ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
  CLIETTE "DTW_SQL:MYDBASE"
ENVIRONMENT (DTW_SYB)      DTWSYB    ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_ORA)      DTWORA    ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_ODBC)     DTWODBC   ( IN DATABASE, LOGIN, PASSWORD,
  TRANSACTION_SCOPE, SHOWSQL, ALIGN, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_APPLET)   DTWJAVA   ( )
ENVIRONMENT (DTW_JAVAPPS)   ( OUT RETURN_CODE ) CLIETTE "DTW_JAVAPPS"
ENVIRONMENT (DTW_PERL)     DTWPERL   ( OUT RETURN_CODE )
ENVIRONMENT (DTW_REXX)     DTWREXX   ( OUT RETURN_CODE )
ENVIRONMENT (DTW_SYSTEM)   DTWSYS    ( OUT RETURN_CODE )
ENVIRONMENT (HWS_LE)       DTWHWS    ( OUT RETURN_CODE )
```

Each ENVIRONMENT statement must be on a single line.

Configuring Live Connection

Live Connection manages database and Java application connections to improve performance for Net.Data on the Windows NT, OS/2, and UNIX operating systems. Through the use of a Connection Manager and cliettes, processes that maintain open connections, Live Connection eliminates the start up overhead of connecting to a database or starting a Java Virtual Machine.

Live Connection uses a configuration file, `dtwcm.cnf`, to determine what cliettes need to be started. It contains administration information and definitions for each of the cliettes used with Live Connection. See “Improving Performance with Connection Management” on page 130 to learn more about Live Connection.

The sample configuration file shown in Figure 6 on page 19 contains the following types of information:

- Connection Manager port information
- SQL cliette information for a DB2 connection

- Java application cliette information

```
 1  CONNECTION_MANAGER{
 2  MAIN_PORT=7100
 3  ADMIN_PORT1=7101
 4  ADMIN_PORT2=7102
 5  }
 6
 7  CLIETTE DTW_SQL:CELDIAL{
 8  MIN_PROCESS=1
 9  MAX_PROCESS=5
10  START_PRIVATE_PORT=7200
11  START_PUBLIC_PORT=7210
12  EXEC_NAME=./dtwcdb2
13  DATABASE=CELDIAL
14  LOGIN=marshall
15  PASSWORD=stlpwd
16  }
17
18  CLIETTE DTW_JAVAPPS{
19  MIN_PROCESS=1
20  MAX_PROCESS=5
21  START_PRIVATE_PORT=7300
22  START_PUBLIC_PORT=7310
23  EXEC_NAME=./javaapp
24  }
```

- Lines 1 - 5 are required for the configuration file and define unique port numbers used with Live Connection.
- Lines 7 - 16 define all database cliettes, identifying the cliette name, the number of processes to be run, database name, port numbers, and the cliette exec file. You can include additional information, such as a user ID and password for connecting to a DB2 database. These additional values are shown in lines 13 - 15.
- Lines 19 - 25 define all cliettes for Java applications, identifying the cliette name, the number of processes to be run, unique port numbers, and the cliette exec file.

Figure 6. The Live Connection configuration file

Before you begin: Read the hints and tips section following these steps before customizing the Live Connection configuration file.

To configure Live Connection ports:

1. Open the configuration file, `dtwcm.cnf`, with an editor.
2. Configure the three Live Connection port numbers:
   - MAIN_PORT
   - ADMIN_PORT1
   - ADMIN_PORT2

   Figure 6 displays the default port numbers. If these numbers are not unique, you must change them to unique port numbers.
3. **Important:** Ensure that the value of MAIN_PORT matches the value of DTW_CM_PORT in the Net.Data initialization file.

To configure the database cliettes:

1. Type the cliette environment statement.

   `CLIETTE` *`type:db_name`*

   Parameters:

   type The name that associates a language environment with a cliette. See on page 39 for a list of valid types.

   db_name
   The database cliette name, which is often the same as the database with which the cliette is associated, such as MYDBASE; however the *db_name* can also be another name. *db_name* is optional when using the Oracle language environment.

2. Determine values for MIN_PROCESS and MAX_PROCESS. MIN_PROCESS specifies the number of processes to be started when the Connection Manager is started. Afterwards, if additional simultaneous requests arrive, the Connection Manager starts more cliettes, adding one as needed, until the value specified for MAX_PROCESS is reached. The values you use can affect performance, but you can change them later.

   Type the MIN_PROCESS and MAX_PROCESS statements:

   ```
   MIN_PROCESS=min_num
   MAX_PROCESS=max_num
   ```

   Parameters:

   min_num
   The number of cliette processes to be started when the Connection Manager is started. You must have enough available unique port numbers for this number of cliettes.

   max_num
   The maximum number of cliettes that can be run simultaneously. You must have enough available unique port numbers for this number of cliettes.

3. Determine which port numbers to use on your system for the database cliette. These numbers must be unique to avoid conflicting with port numbers used for the Cache Manager or other applications. Each cliette uses two ports. When you specify a set of ports, you must specify the range of port numbers to be used. The first two values are START_PUBLIC_PORT and START_PRIVATE_PORT. The other is MAX_PROCESS, indicating the maximum number of cliettes. The following example shows which port numbers are to be used.

   ```
   START_PUBLIC_PORT=1000
   START_PRIVATE_PORT=1010
   MAX_PROCESS=5
   ```

   The example uses the following ports:

1000	1010
1001	1011
1002	1012
1003	1013
1004	1014

   A common error is to have two sets of cliettes overlap the port numbers they use, or overlap with the Cache Manager port numbers. Check with your system administrator to ensure that the port numbers you plan to use are available. The README file for your operating system has general guidelines on what port numbers are valid for your operating system.

4. Specify the name of the cliette executable file. This file name is specified as:

   ```
   EXEC_NAME=./dtwcxxx
   ```

   Where *xxx* is the database type identifier. Refer to Table 4 on page 21 for valid executable file names:

Table 4. Cliette exec file names

Cliette Description	Cliette Type	Names		Platform Availability					
		UNIX	Windows NT or OS/2	AIX	NT	OS/2	HP	SUN	SCO
DB2 process cliette	DTW_SQL	dtwcdb2	dtwcdb2.exe	Y	Y	Y	Y	Y	N
ODBC process cliette	DTW_ODBC	dtwcodbc	dtwcodbc.exe	Y	Y	N	N	N	N
Sybase process cliette	DTW_SYB	dtwcsyb	dtwcsyb.exe	Y	Y	N	N	N	N
Oracle process cliette	DTW_ORA	dtwcora	dtwcora.exe	Y	Y	N	N	N	N

5. Specify the name of the database with which the cliette is associated:

   ```
   DATABASE=db_name
   ```

   Where *db_name* is the name of the database with which the cliette is associated; for example, MYDBASE.

6. Optional: Change the default values for the LOGIN and PASSWORD variables so that Net.Data uses the same user ID that started the Connection Manager to connect to the DB2 database. By specifying these default values, you avoid placing this information in the configuration file. For example, replace lines 14 and 15, in the sample configuration file in Figure 6 on page 19 with these lines:

   ```
   LOGIN=*USE_DEFAULT
   PASSWORD=*USE_DEFAULT
   ```

   Tip: If you define multiple cliette entries in the configuration file, you can specify various database login and passwords for a particular database.

To configure the Java application cliettes:

1. Type the cliette environment statement:

   ```
   CLIETTE DTW_JAVAPPS
   ```

2. Determine values for MIN_PROCESS and MAX_PROCESS. MIN_PROCESS specifies the number of processes that are to be started when the Connection Manager is started. Afterwards, if simultaneous processes arrive, the Connection Manager starts more cliettes, adding one as needed, until the value specified for MAX_PROCESS is reached. The values you use can affect performance, but you can change them later.

   Type the MIN_PROCESS and MAX_PROCESS statements.

   ```
   MIN_PROCESS=min_num
   MAX_PROCESS=max_num
   ```

   Parameters:

   min_num

   The number of cliette processes started when the Connection Manager is started. You must have enough available unique port numbers for this number of cliettes.

max_num
The maximum number of additional cliettes that can be run simultaneously. You must have enough available unique port numbers for this number of cliettes.

3. Determine which port numbers to use on your system for the database cliette. These numbers must be unique to avoid conflicting with port numbers used for the Cache Manager or other applications. Each cliette uses two ports. When you specify a set of ports, you must specify the range of port numbers to be used. The first two values are START_PUBLIC_PORT and START_PRIVATE_PORT. The other is MAX_PROCESS, indicating the maximum number of cliettes. The following example shows which port numbers are to be used.

```
START_PUBLIC_PORT=1000
START_PRIVATE_PORT=1010
MAX_PROCESS=5
```

The example uses the following ports:

1000	1010
1001	1011
1002	1012
1003	1013
1004	1014

A common error is to have two sets of cliettes overlap the port numbers they use, or overlap with the Cache Manager port numbers. Check with your system administrator to ensure that the port numbers you plan to use are available. The README file for your operating system has general guidelines on what port numbers are valid for your operating system.

Hints and tips for configuring Live Connection:

- Cliette names are used by the Connection Manager to uniquely identify a set of cliettes.
- For database cliettes, you must have one named set of cliettes for each database you plan to access. For databases that are rarely accessed, you can set the MIN and MAX number of cliettes to 1. Alternatively, you can also set MIN to 0, which means processes are not started until a Net.Data request is made for the cliette.
- The NAME of the cliette must be consistent with the cliette name referenced in the ENVIRONMENT statement for the cliettes type in initialization file. The cliette name can contain variables, and in the case of DB2 cliettes, it should include the variable reference $(DATABASE). The default value for the cliette name in the ENVIRONMENT statement is DTW_SQL:$(DATABASE). You can use a variable reference in the INI file, but not the Live Connection configuration file.

  The DATABASE variable is defined in the Net.Data macro file. When an SQL statement in the macro file is encountered, the $(DATABASE) variable reference in the Net.Data initialization file is replaced with the current value of DATABASE.

  You can use this method to access multiple databases. If you have three databases that you wanted to access in your Net.Data macro (for example, D1, D2, and D3), and your initialization file has the standard CLIETTE ″DTW_SQL:$(DATABASE)″ line, then you need three sections in the configuration file such as:

```
CLIETTE DTW_SQL:D1{ ...}
CLIETTE DTW_SQL:D2{....}
CLIETTE DTW_SQL:D3{....}
```

- Processes are started but not stopped. If you set the maximum number of processes to M and at any time M processes are used simultaneously, they stay active until you shut down the Connection Manager, therefore you do not want the value of MAX_PROCESS to be so high that you use up all of your system resources starting processes that are rarely used.

  Recommendation: Try using different values for MIN_PROCESS and MAX_PROCESS to see what works best for your system. If the Connection Manager receives more requests than the specified maximum value, the last request is queued until a cliette finishes processing. When a cliette becomes available, the queued request is then processed. This process of queuing requests is transparent to the application user.
- You can use the same type of cliette for different named sections. For example, all DB2 database sections of the configuration file use the same cliette type. You cannot have two sections with the same name.

If you are using CGI, and want only some databases to use Live Connection, simply list the databases you want in the configuration file. When Net.Data is processing a Net.Data macro and encounters an SQL section, it asks the Connection Manager for a specific cliette. If the Connection Manager does not have that type of cliette, it responds with a NO_CLIETTE_AVAIL message. Net.Data processes the request with a DLL version instead.

To automatically start Connection Manager as a Windows NT service:

On Windows NT, you can specify to have Connection Manager start as an Windows NT service, instead of from the command line. Running Connection Manager as an Windows NT service allows Connection Manager to be automatically started each time the machine is started.

Tip: Start Connection Manager from the command line before setting it up to start automatically to insure that the Live Connection configuration file is correct.

- From the Windows NT task bar, select **Start->Settings->Control Panel ->Services**.
- Select **Net.Data Connection Manager** and then click the **Startup** button.
- Select **Automatic startup type** and then click on **OK**.

Configuring Net.Data for FastCGI

FastCGI allows Net.Data can run in FastCGI mode on Apache Web Server and Domino Go Webserver, the follow-on product to IBM Internet Connection Secure Server (ICSS). The FastCGI mode provides similar performance of the other Web API programs with the reliability of CGI-BIN programs (separated memory space).

Before You Begin:

Before you use FastCGI, ensure that you have installed the prerequisite products:

- **For Apache:** Download and install the Apache Web Server 1.2.0 or higher.
- **For Domino Go Webserver:** Download and install the Domino Go Webserver for AIX from:

  `http://www.ics.raleigh.ibm.com/dominowebserver`

To configure Net.Data for FastCGI:

1. Configure the Web server and FastCGI configuration file for your operating system:

   For Apache Web server:

   Update the `http.conf` file.

   - Declare the new application:

     ```
     AppClass inst_dir
     -processes proc_num
     -initial-env LIBPATH=libpath
     -initial-env ORACLE_HOME=oracle_path
     -initial-env ORACLE_SID=oracle_instance
     -initial-env SYBASE=sybase_path
     -initial-env DSQUERY=sybase_instance
     -initial-env DB2INSTANCE=db2_instance
     -initial-env RXQUEUE_OWNER_PID=REXX_perf_var
     -initial-env LANG=locale
     ```

   - Declare the FastCGI module:

     ```
     <location /fcgi-bin>
     SetHandler fastcgi-script
     </location>
     ```

   For Domino Go Webserver on AIX:

   Update the `httpd.conf` and `fcgi.conf` files:

   - In the `httpd.conf` file, declare the service section:

     ```
     ServerInit /u/mydir/http/fcgi-bin/fcgi.o:FCGIInit
     /u/mydir/http/fcgi.conf service/fcgi-bin/*
     /u/mydir/http/fcgi-bin/fcgi.o:FCGIDispatcher*ServerTerm
     /u/mydir/http/fcgi-bin/fcgi.o:FCGIStop
     ```

   - In the `fcgi.conf` file, declare the application:

     ```
     Local {
     Exec inst_dir
     Role Responder
     URL /fcgi-bin/db2www
     BindPath /tmp/db2www.ibm
     NumProcesses proc_num
     Environ LIBPATH=libpath
     Environ ORACLE_HOME=oracle_path
     Environ ORACLE_SID=oracle_instance
     Environ SYBASE=sybase_path
     Environ DSQUERY=sybase_instance
     Environ DB2INSTANCE=db2_instance
     Environ RXQUEUE_OWNER_PID=REXX_perf_var
     Environ LANG=locale
     }
     ```

   Parameters:

   inst_dir

   The path and directory name for Net.Data's executable files.

   For Apache:

   ```
   AppClass /u/mydir/apache/fcgi-bin/db2www
   ```

   For Domino Go Webserver:

   ```
   Exec /u/mydir/http/fcgi-bin/db2www
   ```

   Role Responder

   Required keyword for Domino Go Webserver, only.

   URL Required keyword and URL address for Domino Go Webserver, only. The URL points to the path specified for the EXEC_PATH statement.

BindPath
Required keyword and path statement for Domino Go Webserver on AIX only. The path of the unique UNIX socket used by Net.Data and FastCGI.

proc_num
The number of requests that can be handled simultaneously. The default is 1, but should be increased to improved performance, based on your application requirements. See “Improving Performance with FastCGI” on page 129 for tuning information.

For Apache:

```
-processes 7
```

For ICS or Domino Go Webserver:

```
NumProcesses 7
```

libpath The LIBPATH (shared library or DLL) statements declared in each ENVIRONMENT statement in the Net.Data INI file.

For Apache:

```
-initial-env LIBPATH=/u/mydir/apache/lib:/u/mydir/apache:/usr/lib
```

For Domino Go Webserver:

```
Environ LIBPATH=/u/mydir/http/lib:/u/mydir/http:/usr/lib
```

oracle_path
Required when using Oracle. The path and directory of the Oracle database executable files.

For Apache:

```
-initial-env ORACLE_HOME=/home.native/oracle/product/7.2
```

For Domino Go Webserver:

```
Environ ORACLE_HOME=/home.native/oracle/product/7.2
```

oracle_instance
Required when using Oracle. The instance of the Oracle database. You must use Live Connection for Oracle.

For Apache:

```
-initial-env ORACLE_SID=mvpdb2
```

For Domino Go Webserver:

```
Environ ORACLE_SID=mvpdb2
```

sybase_path
Required when using Sybase. The path and directory of the Sybase database executable files.

For Apache:

```
-initial-env SYBASE=/home.native/sybase/product
```

For Domino Go Webserver:

```
Environ SYBASE=/home.native/sybase/product
```

sybase_instance
Required when using Sybase. The instance of the Sybase database. You must use Live Connection for Sybase

For Apache:

```
-initial-env DSQUERY=SybaseAIX
```

For Domino Go Webserver:

```
Environ DSQUERY=SybaseAIX
```

db2_instance
Required when using DB2. The instance of the DB2 database.

For Apache:

```
-initial-env DB2INSTANCE=wwwinst
```

For Domino Go Webserver:

```
Environ DB2INSTANCE=wwwinst
```

REXX_perf_var
Required when using REXX on AIX. The performance variable is used with FastCGI and REXX on the AIX operating system. The default is 0. For other products and operating systems, declare this variable in the Net.Data macro file. See the "Net.Data for AIX" appendix in *Net.Data Reference* for more information about this variable.

For Apache:

```
-initial-env RXQUEUE_OWNER_PID=0
```

For Domino Go Webserver:

```
Environ RXQUEUE_OWNER_PID=0
```

locale The UNIX locale variable. Use En_US for U.S. English.

For Apache:

```
-initial-env LANG=En_US
```

For Domino Go Webserver:

```
Environ LANG=En_US
```

2. **For Apache:** Add the `fgi-bin` directory as a new script alias in the srm.conf file:
   `ScripAlias /fcgi-bin/ /u/mydir/apache/fci-bin`
3. Migrate any hyperlinks in static or dynamically generated Web pages from CGI-BIN to FCGI-BIN. For example:
   `<A HREF="http://server/fcgi-bin/db2www/filename.ext/block/[?name=val&...]">any text</A>`
4. Modify end-user documentation for URL invocations of Net.Data with FCGI-BIN instead of CGI-BIN. For example:
   `http://server/fcgi-bin/db2www/filename.ext/block/[?name=val&...]`

Configuring Net.Data for Use with the Web Server APIs

Using a Web server application programming interface (API) rather than CGI can improve the performance of Net.Data considerably. Net.Data supports the following server APIs:

- IBM Internet Connection Server API (ICAPI)
- Lotus Domino Go Webserver API (GWAPI)
- Microsoft Internet Server API (ISAPI)
- Netscape API (NSAPI)

For more information about each API, see “Improving Performance using the Web Server APIs” on page 127 and the README file for your version of Net.Data.

Requirement: To run Net.Data in ICAPI, GWAPI, ISAPI, or NSAPI mode, you must reconfigure your Web server to use Net.Data DLLs or shared libraries as its service directives. After reconfiguring, you must restart your Web server so that any changes you make to the Net.Data initialization file take effect. By default, Net.Data runs in CGI mode.

The following sections describe how to configure Net.Data and the Web server to run Web server API mode. General steps and examples are provided, but they might differ for your operating system. Refer to the Net.Data README file for your operating system for specific instructions.

To configure ICAPI and GWAPI:

The Domino Go Webserver is the follow-on product to IBM Internet Connection Secure Server. If you are upgrading, you might wish to use the new Domino Go Webserver. Note that GWAPI and ICAPI are the same product, just renamed to identify which Web server is being used.

1. Stop the Web server.
2. Ensure that the ICAPI or GWAPI DLL or shared library is in the server’s `CGI-BIN` or `ICAPI-LIB` directory.

   See the Net.Data README file or program directory for your operating system for specific file and directory names.
3. Add a service statement to your Web server’s configuration file (`httpd.conf` or `httpd.cnf`) to call the API.

   For example:

   ```
   Service /cgi-bin/db2www*   /usr/lpp/netdata/icapi-lib/db2www:dtw_icapi*
   ```

   See the Net.Data README file for your operating system for specific file and directory names.
4. Restart the Web server.

ICAPI and GWAPI have the full compatibility to support the existing applications. Use the same methods as you use for CGI to invoke a URL, form, or link with ICAPI or GWAPI. Any macro that executes successfully using CGI will execute successfully using ICAPI or and GWAPI. No modifications need to be made to these macros.

To configure ISAPI:

1. Stop the Web server.
2. Copy the DLL for ISAPI that comes with Net.Data into the server’s subdirectory. For example:

   `/inetsrv/scripts/dtwisapi.`*`filetype`*

   Where *filetype* is `.dll` for Window NT and OS/2 and `.o` for UNIX operating systems.

   See the Net.Data README file for your operating system for specific file and directory names.

3. Because ISAPI bypasses CGI processing, you do not need to have the `cgi-bin/db2www/` part of the URL in forms and links. Instead, use `dtwisapi.`*`filetype`*. For example, if the following URL invokes Net.Data as the CGI program:

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/test1.d2w/report
   ```

   Then you should invoke Net.Data as the ISAPI plug-in with the following URL:

   ```
   http://server1.stl.ibm.com/scripts/dtwisapi.dll/test1.d2w/report
   ```

4. If you stored your macro `test1.d2w` in the subdirectory `/order/` under one of the directories specified in MACRO_PATH or current directory of the Web server, invoke Net.Data in CGI mode using the following URL:

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/orders/test1.d2w/report
   ```

   Then the equivalent URL to invoke Net.Data in ISAPI mode is:

   ```
   http://server1.stl.ibm.com/scripts/dtwisapi.dll/orders/test1.d2w/report
   ```

5. Restart the Web server.

To configure NSAPI:

1. Stop the Web server.
2. Copy the DLL for NSAPI that comes with Net.Data into the server directory. For example:

   ```
   /netscape/server/bin/httpd/dtwnsapi.filetype
   ```

   Where *filetype* is `.dll` for Window NT and OS/2 and `.o` for UNIX operating systems.

   See the Net.Data README file for your operating system for specific file and directory names.
3. Modify your server configuration file with the changes listed below. See the Net.Data README file or program directory for your operating system for operating system differences.

obj.conf	Add to the top of the file: `Init fn="load-modules" shlib="<path>dtwnsapi.dll" funcs=dtw_nsapi`

obj.conf	Add to the Services directive:
	`Service fn="dtw_nsapi" method=(GET|HEAD|POST)`
	`  type="magnus-internal/d2w"`

mime.types	Add this type, where *d2w* is the default extension of the macro file. You can specify any three-character combination. `type=magnum-internal/d2w exts=d2w`

4. Move the Net.Data macro files from the `netdata/macro` directory to the server's root document directory:

   ```
   /netscape/server/docs/
   ```

5. Add the server's root document directory to the MACRO_PATH statement, in the initialization file. This change tells Net.Data where to look for the macro files.
6. Because NSAPI bypasses CGI processing, you do not need to have the `cgi-bin/db2www/` part of the URL in forms and links. The server knows files with a `d2w` file type are Net.Data macros because you defined it when you changed the Netscape configuration files. For example, the following URL invokes Net.Data as the CGI program:

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/test1.d2w/report
   ```

While the following URL invokes Net.Data as the NSAPI plug-in:

```
http://server1.stl.ibm.com/test1.d2w/report
```

7. Restart the Web server.

If you keep your Net.Data macros in several directories, the last three steps change:

1. Move the directories with the Net.Data macros they contain to the server's root document directory.
2. Update the MACRO_PATH variable in the initialization file to include all of the directories and subdirectories where you macro files are located.
3. Modify the links and forms that point to these Net.Data macros, keeping their directory names. For example, when running in CGI mode, the following URL calls a Net.Data macro that is stored in the `/orders/` directory:

   ```
   http://server1.stl.ibm.com/cgi-bin/db2www/orders/test1.d2w/report
   ```

   The updated URL used to invoke Net.Data in NSAPI mode is shorter, but keeps the directory name:

   ```
   http://server1.stl.ibm.com/orders/test1.d2w/report
   ```

Configuring Net.Data with the Net.Data Administration Tool

The Net.Data administration tool helps you to configure and manage the Net.Data initialization file (DB2WWW.INI) and the configuration file for Live Connection (DTWCM.CNF) on the Windows NT, AIX, and OS/2 operating systems. Using this tool, you can complete the following tasks:

- "Starting the Administration Tool" on page 30
- "Configuring Path Statements" on page 30
- "Configuring Ports" on page 32
- "Configuring Cliettes" on page 32
- "Configuring Language Environments" on page 36
- "Defining Configuration Variables" on page 39

See "Before You Begin" to learn about setting up the administration tool and ensuring your have the correct software prerequisites.

Before You Begin

1. Plan the configuration of Net.Data language environments, databases, cliettes, ports, and configuration variables.
2. Install Net.Data from CD-ROM.
3. Install the Java run-time libraries (JDK 1.1 and subsequent versions for each operating systems). Check the Net.Data README file for your operating system for more information.

   Make sure you have `classes.zip` in your CLASSPATH after installing JDK.
4. If you have installed the IBM JDBC driver that is packaged with DB2 Universal Database, add the driver directory to your Java CLASSPATH statement to enable the DB2 login test.
5. Change to the directory where the Net.Data administration tool program is stored:

For OS/2 and Windows NT:

The *inst_dir*\connect*admin_directory*, where *inst_dir* is the directory you specified for Net.Data during installation and *admin_directory* is the directory where the administration tool files exist.

For AIX:

The /usr/lpp/internet/db2www/db2.v2/*admin directory*, where *admin_directory* is the directory where the administration tool files exist

Starting the Administration Tool

The operating system that you use determines how you start the administration tool.

For OS/2 and Windows NT:

From the IBM Net.Data Version 2 folder, select the **Net.Data Admin Tool** icon.

For AIX:

Change to the Net.Data installation directory (inst_dir). From the command line, enter ndadmin to start the tool.

The administration tool is launched and the Net.Data Administration notebook is displayed.

Configuring Path Statements

Use the **Path** page to add, modify, or delete the path statements for locating the files the Net.Data needs to process Net.Data macros. These statements are described in “Customizing Path Configuration Statements” on page 13. Figure 7 on page 31 shows the **Path** page.

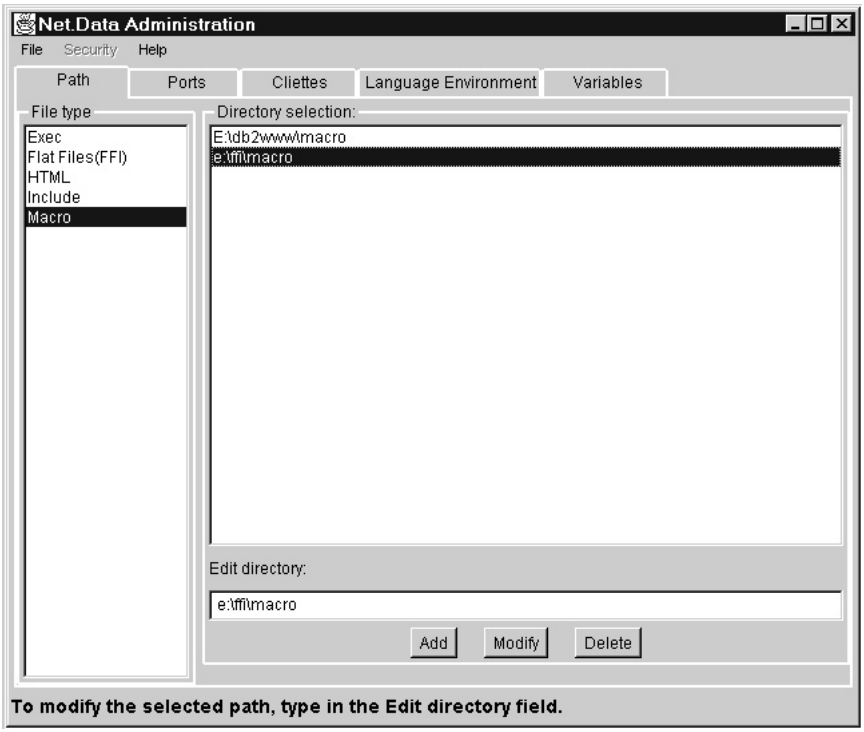


Figure 7. The Path Page of the Net.Data Administration Tool. Use this page to add, modify or delete path statements.

Configuration tip: The HTML file type can have one path, only.

To add a path statement:

1. Start the administration tool.
2. From the **Path** page, select a file type from the **File type**, for example, select `Exec`.
3. In the **Edit directory** field, type the new path and click on the **Add** button.

   If the path you specified does not exist, a warning window opens. If no directory is selected, the new directory is added as the last item in the list.
4. Close the administration tool, or click on another tab to complete additional configuration tasks.

To modify a path statement:

1. Start the administration tool.
2. From the **Path** page, select the file type you want to change from the **File type** list.
3. Select the path you want to modify in the **Directory selection** list. The selected path opens in the **Edit directory** field.
4. Modify the path in the **Edit directory** field and click on the **Modify** button. If the path you entered does not exist, a warning window opens.
5. Close the administration tool, or click on another tab to complete additional configuration tasks.

To delete a path statement:

1. Start the administration tool.

2. From the **Path** page, select the file type that you want to delete from the **File type** list.
3. In the **Directory selection** field, select the path you want to delete. The selected path opens in the **Edit directory** field.
4. Click on the **Delete** button.
5. Close the administration tool, or click on another tab to complete additional configuration tasks.

Configuring Ports

Use the **Port** page to specify the TCP/IP port numbers used by Net.Data. Figure 8 shows the **Port** page.

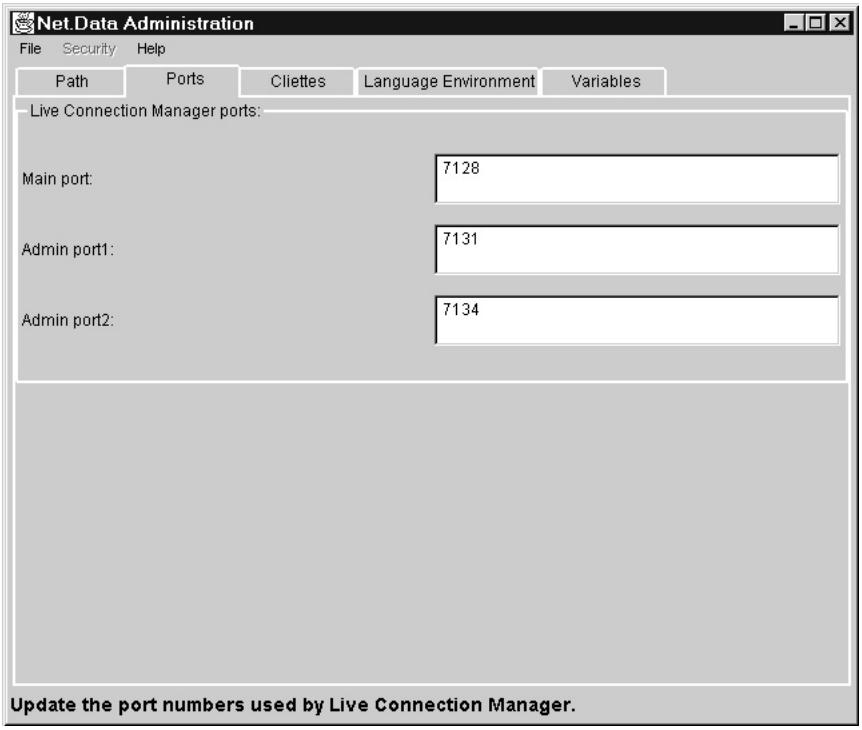


Figure 8. The Port Page of the Net.Data Administration Tool. Use this page to specify ports.

To specify TCP/IP port numbers:

1. Start the administration tool.
2. From the **Port** page, type a unique port number in each of the port fields. The administration tool verifies the port number you type in each field when you tab to the next field.
3. Close the administration tool, or click on another tab to complete additional configuration tasks.

Configuring Cliettes

Use the **Cliette** page to add, modify, or delete Live Connection database cliettes, and you can also manage database and administrator user IDs and passwords for database cliettes. More information about cliettes is provided in “Improving

Performance with Connection Management” on page 130. Figure 9 shows the **Cliette** page.

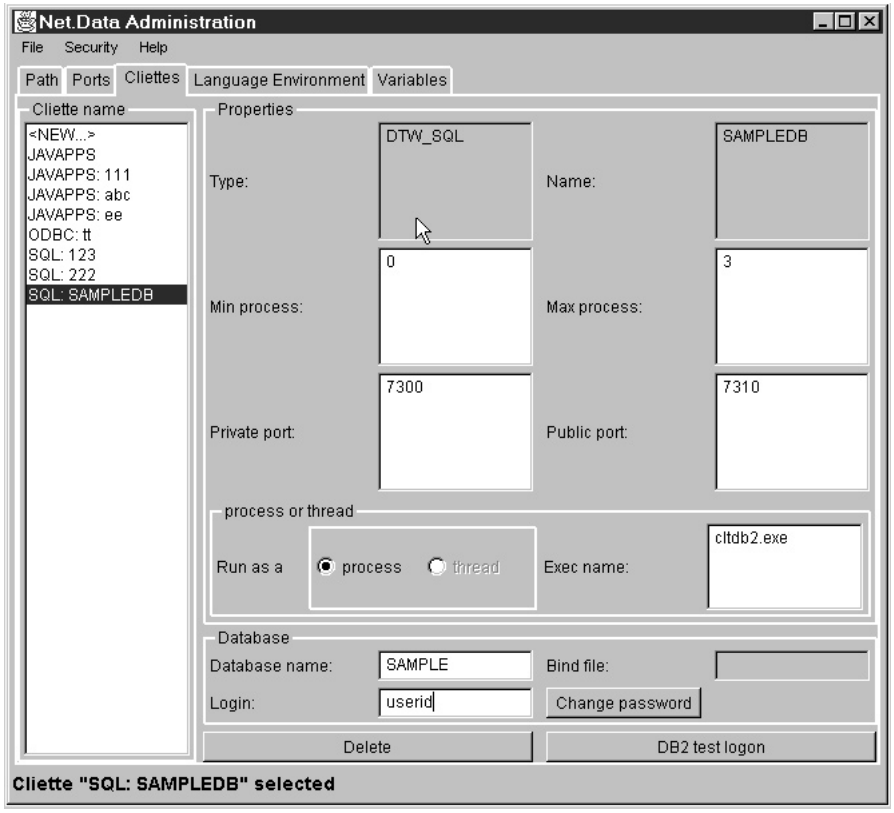


Figure 9. The Cliette Page of the Net.Data Administration Tool. Use this page to add, modify, and delete cliettes.

To add a cliette:

1. Start the administration tool.
2. From the **Cliette** page, select **<NEW...>** from the **Cliette name** list. The **Add a cliette** window opens.

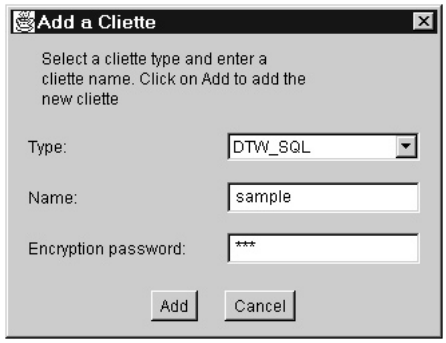


Figure 10. The Add a Cliette Window of the Net.Data Administration Tool. Use this page to add cliettes.

If you have enabled encryption, you are prompted for the encryption password the first time you create or modify a cliette. This password is saved and you will not ever have to enter it again.

3. Select a cliette type from the **Type** list.
4. Type a name for the new cliette in the **Name** field. The name can be the name of the database or another unique cliette name. For example: MYCLIETTE.
5. Type the encryption password if the **Encryption password** field is enabled. You will not need to type the password again, as the administration tool saves the password for you.
6. Click on the **Add** button.

   The new cliette is created and is added to the bottom of the cliette list. Additionally, the new name is highlighted and the default properties for the cliette are displayed in the **Properties** group box. You can change these values to fit your configuration.
7. Close the administration tool, or click on another tab to complete additional configuration tasks.

To modify a cliette:

1. Start the administration tool.
2. From the **Cliette** page, select the name of the cliette that you want to change from the **Cliette name** list. The properties of the cliette are displayed in the **Properties** group box.
3. Modify the properties from the **Properties** group box, as needed.
   a. The **Type** field displays the type of cliette that is being defined and corresponds to a language environment type name. Net.Data populates this field when you add a new cliette, and the choices are defined in the **Cliette type** list in the Add a Cliette window.
   b. The **Name** field displays the name of the cliette, which is usually the name of the database. Net.Data populates this field when you add a new cliette.
   c. Type the number of cliette process that can be started when Connection Manager is started in the **Min process** field. You need one unique port address for each process. See “Configuring Live Connection” on page 18 for more information about MIN Process values.
   d. In the **Max process** field, type the number of cliette processes that can be run at the same time, in addition to the processes started when Connection Manager is started. You need one unique port address for each process. See “Configuring Live Connection” on page 18 for more information about MAX Process values.
   e. Type a unique port number in the **Private port** field to specify the starting port number for use with the cliette processes that are started with the Connection Manager. An additional port number is used for each of the processes specified by the **Min Process** value. For example, if you specify the port number 7012 for **Private port** and the value 5 for **Min process**, port numbers 7012-7016 are used and must not conflict with other port assignments in the system.
   f. Type a unique port number in the **Public port** field to specify the starting port number used with the cliette processes that are started when additional processes are started, up to the number specified in the **Max process** field. An additional port number is used for each of the processes For example, if you specify the port number 7020 for **Public port** and the value 5 for **Max process**, port numbers 7020-7024 are used and must not conflict with other port assignments in the system.
   g. The **Exec name** field displays the name of the cliette executable file.

4. If the cliette is being used with a database, modify the values for the **Database** group box, as needed:
   a. Specify the database name of the database with which the cliette is associated **Database name** field, for example, MYDBASE.
   b. The **Bind file** field contains the name and path of the bind file for the type of cliette that you are using.
   c. The **Login** field specifies the login user ID used to connect to the database.
   d. The **Change password** push button opens the Change Database Password window. Type the encryption password and the new password, twice. You can encrypt the database password by using the encryption functions specified in the **Security** pull-down menu.
5. Select **File** and then **Save** to save your changes.
6. Close the administration tool, or click on another tab to complete additional configuration tasks.

To test the DB2 database logon and connection:

1. From the **Cliette** page of the administration tool, click on the **DB2 test logon** push button. When the test is complete a confirmation window opens, displaying the status of the connection test.
2. Close the window to continue configuring or close the administration tool.

To delete a cliette:

1. Start the administration tool.
2. From the **Cliette** page, select the name of the cliette that you want to delete from the **Cliette name** list.
3. Click on the **Delete** button.
4. Close the administration tool, or click on another tab to complete additional configuration tasks.

To turn on encryption of cliette user IDs and passwords:

Encryption provides security for database connections with cliettes. When encryption is turned on, all database passwords in the Live Connection configuration file are encrypted and require an encryption password for access and decryption.

Requirement: You must use a Net.Data Version 2 Live Connection configuration file to use encryption.

1. **Important:** Back up a copy of your Live Connection configuration file, `<path>dtwcm.cnf`. You need this file if you lose the encryption password, or want to decrypt database passwords and need to restore the passwords.
2. From the **Cliette** page of the administration tool, select the **Security -> Turn encryption on** pull-down menu option. The Turn Encryption On confirmation window opens.
3. Click on **Yes** to continue. The Encryption Password window opens.
4. Type the password twice for authorization to work with cliettes that have encrypted passwords.
5. Click on **OK** to define the new password and encrypt all of the database passwords for your cliettes.

To turn off encryption of cliette user IDs and passwords:

1. From the **Cliette** page of the administration tool, select the **Security -> Turn encryption off** pull-down menu option. The Turn Encryption Off confirmation window opens.
2. Click on **Yes** to continue. All passwords are set to *USE_DEFAULT for security reasons. You can restore your passwords from the backup copy of the Live Connection file, <path>dtwcm.cnf.

To change the password for encryption:

1. From the **Cliette** page of the administration tool, select the **Security -> Change Encryption Password** pull-down menu option. The Change Encryption Password confirmation window opens.
2. Click on **Yes** to continue. The Change Encryption Password window opens.
3. Type the old encryption password once, and the new password twice.
4. Click on **OK** to change the encryption password.

To change the database password:

1. From the **Cliette** page of the administration tool, click on the **Change Password** push button. The Change Database Password window opens.
2. Type the encryption password once and the new database password twice.
3. Click on **OK** to change the password and close the window. The changed database password is encrypted if you have turned encryption on.

Configuring Language Environments

Use the **Language Environment** page to add, modify, or delete Net.Data language environments. Language environments are discussed in “Environment Configuration Statements” on page 16. Figure 11 on page 37 shows the **Language Environment** page.

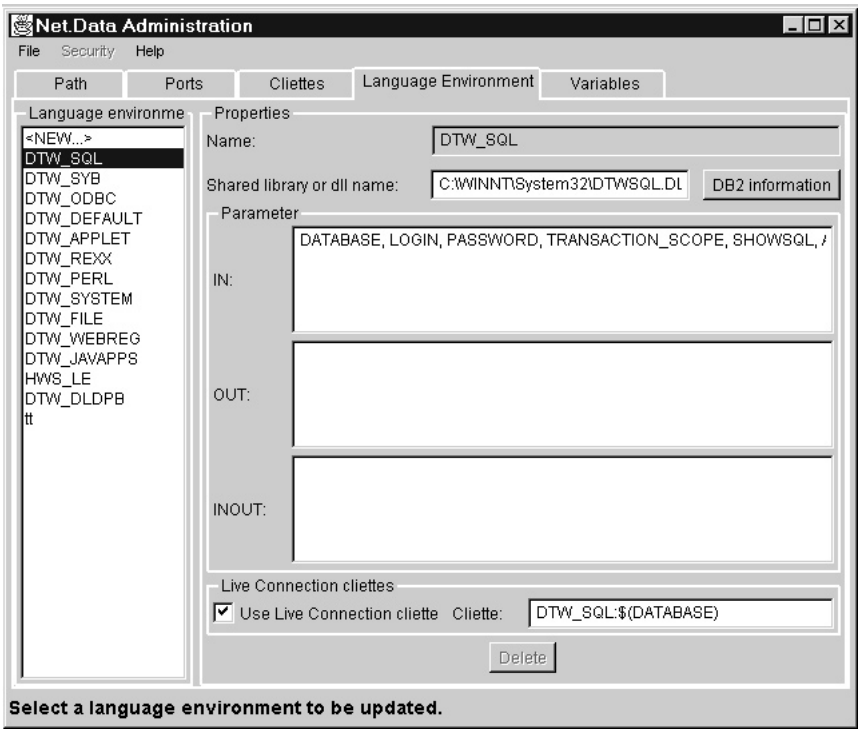


Figure 11. The Language Environment Page of the Net.Data Administration Tool. Use this page to specify language environments.

To add a language environment:

1. Start the administration tool.
2. From the **Language Environment** page, select **<NEW...>** from the **Language environment** list. The **Add a new language environment** window opens.
3. Type a name of the language environment in the field and click on the **Add** button. The Add a Language Environment window opens.

Figure 12. The Add a Language Environment window of the Net.Data Administration Tool. Use this page to specify a new language environment.

   The new language environment is created and its name is added to the bottom of the language environment list. Additionally, the new name is highlighted, and the default properties for the language environment are displayed in the **Properties** group box. You can change these values to fit your configuration.
4. Close the administration tool, or click on another tab to complete additional configuration tasks.

To modify a language environment:

1. Start the administration tool.

2. From the **Language Environment** page, select the name of the language environment that you want to change from the **Language environment** list. The properties of the cliette are displayed in the **Properties** group box.
3. Modify the properties in the **Properties** group box, shown in Figure 12 on page 37 as needed:

   a. Specify the name of the language environment in the **Name** field; this name corresponds to the language environment type used to define a cliette. To change this value, double click on a different name from the **Language environment** list. See "Environment Configuration Statements" on page 16 for more information about language environment types.

   b. Specify the shared library or DLL program name and path for the language environment in the **Shared library or dll name** field.

   c. Select the **DB2 information** push button to display the DB2 Information window as shown in Figure 13.
   Specify the values for the DB2 environment variables:

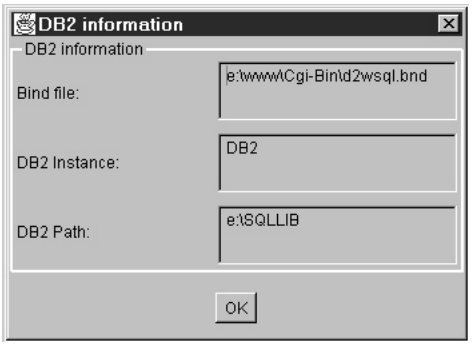


Figure 13. The DB2 Information window of the Net.Data Administration Tool. Use this page to specify information specifically for DB2 databases.

      1) Type the path and file name of the bind file in the **Bind file** field.
      2) Specify the DB2INSTANCE value for the database associated when you use the SQL language environment in the **DB2 Instance** field.
      3) Specify the path directory name for the DB2 product executable files, usually \SQLLIB, in the **DB2 Path** field.
      4) Click on **OK** to save your changes and close the window.

   d. Specify the input and output parameters that are passed to or from a language environment each time the language environment is called in the **Parameters** group box.

      Tip: Do not update these fields unless you are defining your own language environment.

   e. Specify whether to use cliettes and which cliette should be associated with the language environment in the **Live Connection cliettes** group box.

      1) Specify whether the cliette for the language environment is active by checking the **Use Live Connection cliette** check box. Select this check box if you want to use the cliette specified in the **Cliette** field when calling the language environment.
      2) Specify the name of the cliette that is to be run with the language environment being defined in the **Cliette** field. The syntax of the name depends on whether you are configuring a database or the Java Application language environment. The default is `DTW_SQL:$(DATABASE)`.

         Syntax for Databases:

         type`:`*name*

Where:

type The language environment type for the cliette. It can be one of the following values:

For Windows NT:
DTW_ODBC, DTW_ORA, DTW_SYB, DTW_SQL, DTW_JAVAPPS

For OS/2:
DTW_SQL, DTW_JAVAPPS

For AIX:
DTW_ODBC, DTW_ORA, DTW_SYB, DTW_SQL, DTW_JAVAPPS

name The name of the cliette as defined on the **Cliette** page. The default is $(DATABASE).

Syntax for Java Applications:

```
DTW_JAVAPPS
```

4. Select **File** and then **Save** to save your changes
5. Close the administration tool, or click on another tab to complete additional configuration tasks.

To delete a language environment:

Restriction: You can delete only the language environments created by users, not the default language environments that come with Net.Data.

1. Start the administration tool.
2. From the **Language Environment** page, select the name of the language environment that you want to delete from the **Language environment** list.
3. Click on the **Delete** button.
4. Close the administration tool, or click on another tab to complete additional configuration tasks.

Defining Configuration Variables

Use the **Variables** page to specify the home directory for Net.Data and to select the level of error messages logging. Figure 14 on page 40 shows the **Variables** page.

Figure 14. The Variables Page the Net.Data Administration Tool. Use this page to specify initialization variables.

To specify the home directory for Net.Data:

This variable is also known as the installation directory variable.

1. Start the administration tool.
2. From the **Variables** page, type the path for the directory where the log file is to be stored in the **Installation directory** field. The default is `\inst_dir\logs\`. For example: `e:\db2www`.
3. Close the administration tool, or click on another tab to complete additional configuration tasks.

To specify the error message logging level for Net.Data:

1. Start the administration tool.
2. From the **Variables** page, select a level of error logging from the **Error logging** group box:
   - **off**
   - **errors only**
   - **both warnings and errors**
3. Close the administration tool, or click on another tab to complete additional configuration tasks.

Specifying Access Rights to Net.Data Files

Before using Net.Data, you need to ensure that the user IDs under which Net.Data executes have the appropriate access rights to files used by Net.Data. This means that these files must be in directories or libraries to which the Web server can connect, or to which these user IDs have explicit access rights.

More specifically, ensure that the user ID under which Net.Data executes have the following authorizations:

- To read the Net.Data initialization file, `db2www.ini`
- To execute the Net.Data executables and DLLs, and to search the directories in the paths to the executables and DLLs
- To read the appropriate Net.Data macro files and search the appropriate directories identified by the MACRO_PATH path configuration statement
- To execute the appropriate files and to search the appropriate directories identified by the EXEC_PATH path configuration statement
- To read the appropriate files and to search the appropriate directories identified by the INCLUDE_PATH path configuration statement
- To read and write the appropriate files and to search the appropriate directories identified by the FFI_PATH path configuration statement
- To read the Live Connection configuration file, DTWCN.CNF
- To read the Cache Manager configuration file, CACHEMGR.CNF
- To read external Perl and REXX executable files referenced by the language environments

The methods for granting access to these files depend on the operating system on which Net.Data is running.

Chapter 3. Keeping Your Assets Secure

You must decide on what level of security is appropriate for your assets. This chapter describes methods you can use for keeping your assets secure and also provides references to additional resources you can use to plan for the security of your Web site.

The following sections contain guidelines for protecting your assets. The security mechanisms described include:

- "Using Firewalls"
- "Encrypting Your Data on the Network" on page 45
- "Using Authentication" on page 45
- "Using Authorization" on page 45
- "Using Net.Data Mechanisms" on page 45

Additionally, Net.Data provides database cliette password encryption; see "Configuring Cliettes" on page 32 for more information.

Using Firewalls

Firewalls are collections of hardware, software, and policies that are designed to limit access to resources in a networked environment.

Firewalls

- Protect the internal network from infiltration or intrusion
- Protect the internal network from data and programs that are brought in by internal users
- Limit internal user access to external data
- Limit the damage that can be done if the firewall is breached

Net.Data can be used with a firewall product that executes in your environment.

The following possible configurations provide recommendations for managing the security of your Net.Data application. these configurations provide high-level information and assume that you have configured a firewall that isolates your secure intranet from the public Internet. Carefully consider these configurations with your organization's security policies:

- **High security configuration:**

  This configuration creates a subnetwork that isolates Net.Data and the Web server from both the secure intranet and the public Internet. The firewall software is used to create a firewall between the Web server and the public Internet, and another firewall between the Web server and the secured intranet, which contains DB2 Server. This configuration is shown by Figure 15 on page 44.

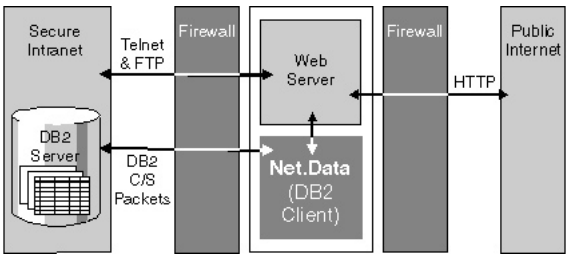


Figure 15. High Security Configuration

To set up this configuration:

- Install Net.Data on the Web server machine and ensure that Net.Data can access DB2 Server inside the intranet by:
  - Installing Client Application Enabler (CAE) on the Web server machine.
  - Configure the firewall to allow DB2 traffic through the firewall. One method is to add a packet filtering rule to allow DB2 client requests from Net.Data and acknowledge packets from DB2 Server to Net.Data.
- Allow FTP and Telnet access between the Web server and the secure intranet. One method is to install a socks server on the Web server machine.
- In the packet filtering configuration file of the firewall software, specify that incoming TCP packets from the standard HTTP port can access the Web server. Also, specify that outgoing TCP acknowledge packets can go to any hosts on the public Internet from the Web server.

- **Intermediate security configuration:**

  In this configuration, firewall software isolates the secured intranet with DB2 server from the public Internet. Net.Data and the Web server are outside the firewall on a workstation platform. This configuration is simpler than the first, but still offers database protection. Figure 16 shows this configuration.

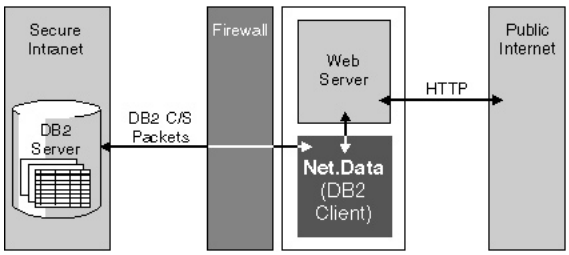


Figure 16. Intermediate Security Configuration:

You must install CAE on the Web server to allow Net.Data to communicate with DB2 server. The firewall must be configured to allow DB2 client requests to flow from Net.Data to DB2 and to allow acknowledge packets to flow from DB2 to Net.Data.

- **Low security configuration:**

  In this configuration, DB2 server and Net.Data are installed outside of the firewall and the secured intranet. They not protected from external attacks. The firewall needs no packet filtering rules for this configuration. Figure 17 on page 45 shows this configuration.

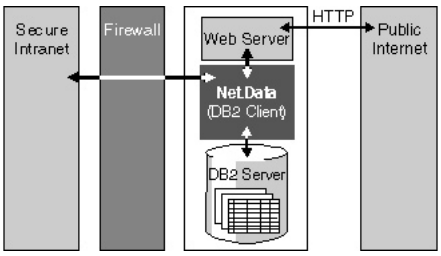


Figure 17. Low Security Configuration:

Encrypting Your Data on the Network

You can encrypt all data that is sent between a client system and your Web server when you use a Web server that supports Secured Sockets Layer (SSL). This security measure supports the encryption of login IDs, passwords, and all data that is transmitted through HTML forms from the client system to the Web server and all data that is sent from the Web server to the client system. Most Web servers support SSL, such as Internet Connection Secure Server, Version 2 Release 2 or higher and Lotus Domino Go Webserver, 4.6.1 or higher.

Using Authentication

The Web server associates a user ID with each Net.Data request that it processes. The process or thread that is handling the request can then access any resource to which that user ID is authorized.

Net.Data supports two types of authentication: one protecting certain directories on your server and one protecting your database.

- Most Web servers allow you to specify directories on the server to protect. You can also have your system require a user ID and password for people accessing files in directories you specify. See the *Administrator's Guide* for your Web server to determine your system's capabilities.
- DB2 has an authentication system for database access that can restrict access to tables and columns to certain users. You can use Net.Data's special variables, such as LOGIN and PASSWORD, to link to the DB2 authentication routine.

Using Authorization

Data sources such as DB2 provide their own authorization mechanisms to protect the information that they manage. These mechanisms assume that the user ID associated with the process that is executing the Net.Data request has been properly authenticated, as explained in"Using Authentication". The existing access control mechanisms for these data sources then either permit or deny access based on the authorizations that are held by the authenticated user ID.

Using Net.Data Mechanisms

In addition to the methods described above, you can use other Net.Data-provided mechanisms such as path statements and hidden variables, as well as methods that use HTML forms or SQL statements.

Net.Data evaluates the settings of path configuration statements to determine the location of files and executable programs that are used by Net.Data macro files. These path statements identify one or more directories that Net.Data searches when attempting to locate macro files, executable files, and include files. By selectively including directories on these path statements, you can explicitly control the files that are accessible by users at browsers. Refer to “Chapter 2. Configuring Net.Data” on page 3 for additional detail about path statements.

You can use hidden variables to conceal various characteristics of your Net.Data macro from users that view your HTML source with their Web browser. For example, you can hide the internal structure of your database. See “Hidden Variables” on page 68 for more information.

You can also use the following methods to set up a protection scheme:

- Create your own protection scheme using Net.Data. For example, you can request validation information from a user through an HTML form and validate it using data in a database or through an external program called from a Net.Data macro.
- Protect your assets by the SQL statements you allow people to send to the database, for example, limiting SELECT statements to two tables.

For more information on protecting your assets, see the Internet security list of frequently asked questions (FAQ) at this Web site:

`http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html`

Chapter 4. Invoking Net.Data

You can configure Net.Data for to use the Common Gateway Interface (CGI) or FastCGI, or with a Web server API like Lotus Domino Go Webserver (GWAPI), Internet Connection Server (ICAPI), Netscape Server (NSAPI), and Microsoft Internet Server (ISAPI). The syntax used to invoke Net.Data depends on how Net.Data is configured. See “Chapter 2. Configuring Net.Data” on page 3 to learn about configuring Net.Data.

This chapter discusses invoking Net.Data with CGI. See “Improving Performance using the Web Server APIs” on page 127 to learn how to invoke Net.Data in API mode.

You can also specify whether you want Net.Data to execute a macro file or only a single SQL statement, stored procedure, or function. These types of invocation are called macro request and direct request, respectively.

Macro Request
Invokes Net.Data by specifying a macro file that is written in the Net.Data macro language.

Direct Request
Invokes Net.Data by specifying:

- The name of a language environment
- An SQL statement or the name of a program, along with any parameter values that are required for the invocation of the function
- Form data that is required for invocation of the SQL statement or function

Web developers who wish to write a single SQL query or call a single function such as a DB2 stored procedure, REXX program or Perl function can now issue a direct request to the database. A direct request does not have any complex Net.Data application logic that requires a Net.Data macro file, and therefore bypasses the Net.Data macro processor. Direct request parameters are passed to the appropriate language environment for processing for improved performance.

Figure 18 on page 48 illustrates the differences between a macro request and a direct request. A macro request always specifies a macro file within the URL for the request and can also use form data. A direct request never specifies a macro file within the URL, but can still use form data.

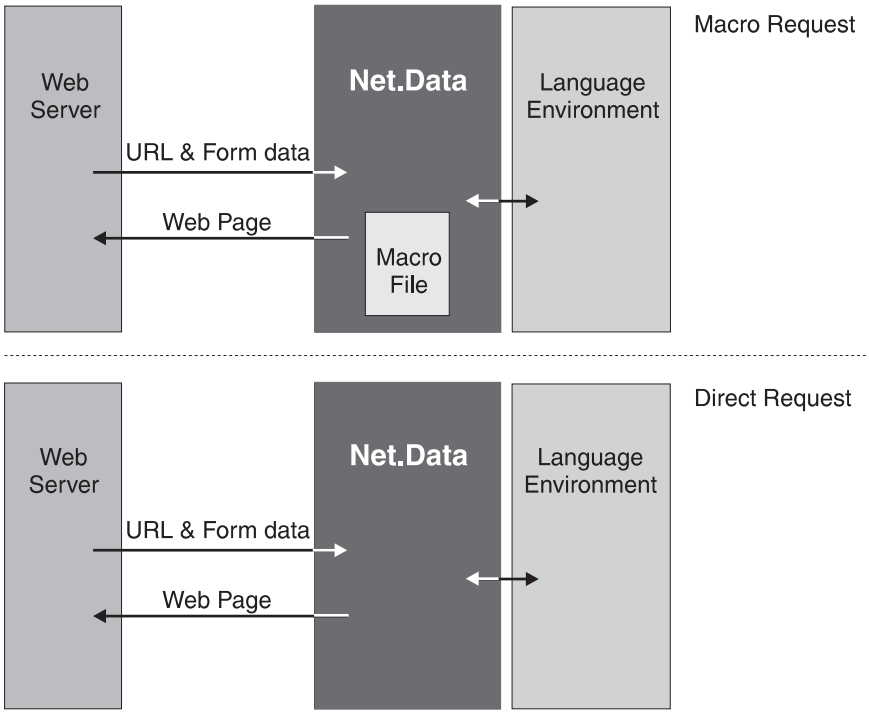


Figure 18. Macro Request Versus Direct Request

The syntax for invoking Net.Data depends on how Net.Data is configured and the type of request that you make. For both macro and direct requests, Net.Data can be invoked from a Web browser by an HTML link, an HTML form, or a URL. The Web server invokes Net.Data using CGI, FastCGI, or one of the Web server APIs.

For macro requests, the name of the Net.Data macro file and the name of the HTML block that is to be executed within the Net.Data macro are specified within the link, form, or URL.

For direct requests, the name of the Net.Data language environment, the SQL statement or the name of the program, and any additional required parameter values are specified within the URL using a syntax defined by Net.Data.

This chapter describes both invocation methods:

- "Invoking Net.Data with a Macro File (Macro Request)"
- "Invoking Net.Data without a Macro File (Direct Request)" on page 51

Invoking Net.Data with a Macro File (Macro Request)

This section shows you how to invoke Net.Data by specifying a macro file. For the macro request style of invocation, you can call Net.Data using a URL, an HTML form, or an HTML link.

The following examples show the different ways you can invoke Net.Data.

- HTML link:

```
<A HREF="http://server/cgi-bin/db2www/filename.ext/block/
[?name=val&...]">any text</A>
```

- HTML form:

  ```
  <FORM METHOD=method ACTION="http://server/cgi-bin/db2www/
  filename/block/[?name=val&...]">any text</FORM>
  ```

- URL:

  ```
  http://server/cgi-bin/db2www/filename/block/[?name=val&...]
  ```

Parameters:

server Specifies the name of the Web server. If the server is the local server, you can omit the server name and use a relative URL.

filename Specifies the name and extension of the Net.Data macro file, where the file name is the relative path under the directory specified by the MACRO_PATH.

block Specifies the name of the HTML block in the referenced Net.Data macro file.

method Specifies the HTML method used with the form. METHOD=POST is recommended.

?name=val&...
Specifies one or more optional parameters passed to Net.Data.

HTML Links

You can create a link in a Web page that results in the execution of an HTML block by using the HTML <a> tag in the macro file. You accomplish this by using the HREF attribute to specify the macro and HTML block, and by including some text or even an image within the link tag. This method identifies the text or image as a “hot spot” when the Web page is displayed at the browser. When a user at a browser clicks on the text or image, Net.Data executes the HTML block within the macro.

The following example shows a link that results in the execution of an SQL query when a user selects the text ″List all monitors″ on a Web page.

```
<a href="http://server/cgi-bin/db2www/listA.d2w/report">
List all monitors</a>
```

The link calls this macro:

```
%DEFINE DATABASE="MNS97"

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

%HTML(report){
@myQuery()
%}
```

The query returns a table that contains model number, cost, and description information for each monitor that is described within the EQPTABLE table. This example displays the results of the query by generating a default report. See “Report Blocks” on page 86 for information on how you can customize your reports using a REPORT block.

Generally, each block of a Net.Data macro begins with `%block_name{` and ends with `%}`. See *Net.Data Reference* for additional detail about the syntax of the Net.Data macro language.

HTML Forms

You can dynamically customize the execution of your Net.Data macros using HTML forms. Forms allow users to provide input values that can affect the execution of the macro and the contents of the Web page that Net.Data builds.

The following example builds on the monitor list example in “HTML Links” on page 49 by letting users at a browser use a simple HTML form to select the type of product for which information will be displayed.

```
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD=POST ACTION="/cgi-bin/db2www/equiplst.d2w/report">
<P>What type of hardware do you want to see?
<MENU>
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="MON" checked> Monitors
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PNT"> Pointing devices
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="PRT"> Printers
<LI><INPUT TYPE="RADIO" NAME="hdware" VALUE="SCN"> Scanners
</MENU>

<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
```

After the user at the browser makes a selection and clicks on the Submit button, the Web server processes the ACTION parameter of the FORM tag, which invokes Net.Data. Net.Data then executes the HTML report block in the `equiplst.d2w` macro:

```
%DEFINE DATABASE="MNS97"

 %HTML(input)%{
%}
 %FUNCTION(DTW_SQL) myQuery(){
SELECT MODNO, COST, DESCRIP FROM EQPTABLE
WHERE TYPE='$(hdware)'
%REPORT{
<H3>Here is the list you requested</H3>
%ROW{
<HR>
$(N1): $(V1), $(N2): $(V2)
<P>$(N3): $(V3)
%}
%}
%}

%HTML(report){
@myQuery()
%}
```

In the above example, the value of `TYPE=$(hdware)` in the SQL statement is taken from the HTML form input.

See *Net.Data Reference* for a detailed description of the variables that are used in the ROW block.

Invoking Net.Data without a Macro File (Direct Request)

This section shows you how to invoke Net.Data without specifying a macro file. For the direct request style of invocation, you specify the name of the Net.Data language environment, the SQL statement or a program to be executed, and any additional required parameter values within the URL, using a syntax defined by Net.Data.

The SQL statement or program and any other specified parameters are passed directly to the designated language environment for processing. Direct request improves performance because Net.Data does not need to read and process a macro file. The SQL, ODBC, Oracle, Sybase, Java, System, Perl, and REXX Net.Data-supplied language environments support direct request invocation, and you can call Net.Data using a URL, an HTML form, or a link.

A direct request invokes Net.Data by passing parameters in the (NAME=VALUE) pair section in the query string of the URL or the form data. The following example illustrates the context in which you specify a direct request.

```
<A HREF="http://server/cgi-bin/db2www?direct_request">any text</A>
```

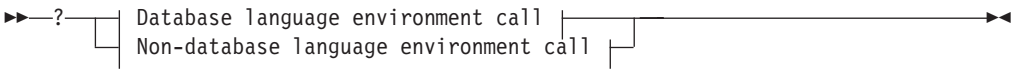
Where *direct_request* represents the direct request syntax. For example, the following HTML link contains the direct request:

```
<A HREF="http://server/cgi-bin/db2www?LANGENV=DTW_PERL&FUNC=my_perl(hi)">
  any text</A>
```

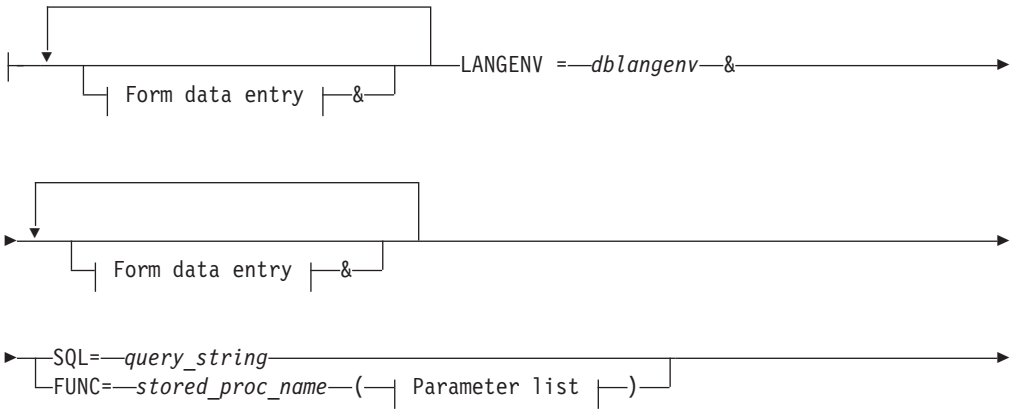
Direct Request Syntax

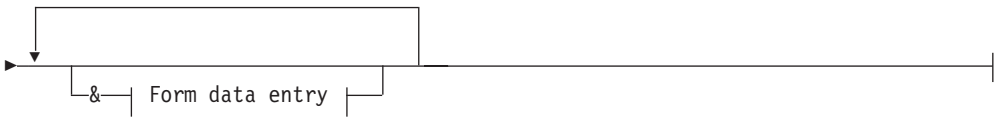
The syntax for invoking Net.Data with direct request can contain a call to either a database or a non-database language environment.

Syntax

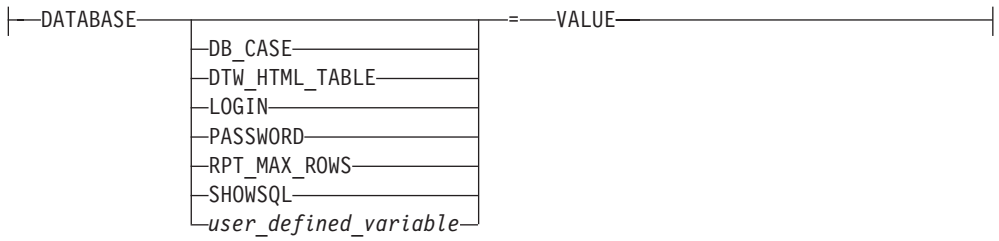


Database language environment call

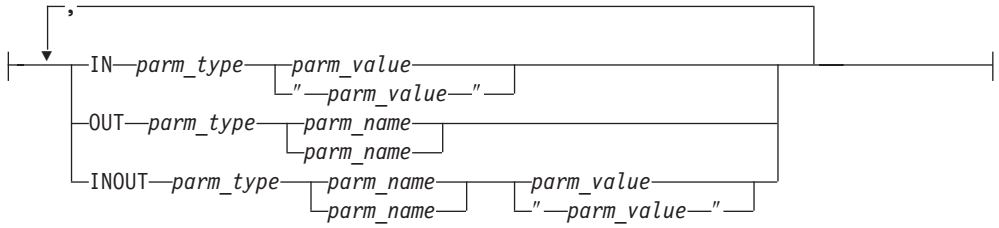




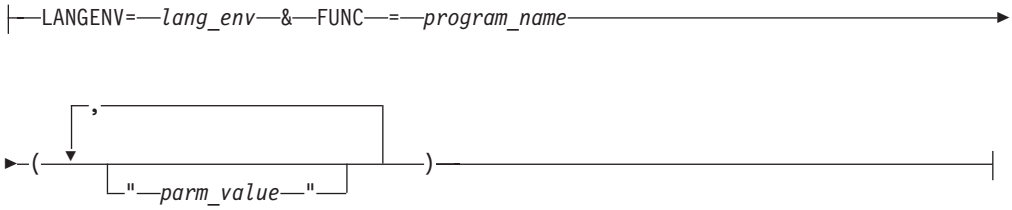
Form data entry



Parameter list



Non-database language environment call



Parameters

Database language environment call

Specifies a direct request to Net.Data that invokes a database language environment.

Form data entry

Parameters that allow you to specify the settings of SQL variables or to request simple HTML formatting. See the variables chapter of *Net.Data Reference* to learn more about these variables.

DATABASE

Specifies the database to which Net.Data should pass the SQL request. This parameter is required.

DB_CASE
Specifies the case (upper or lower) for SQL statements.

DTW_HTML_TABLE
Specifies whether Net.Data should return an HTML table.

LOGIN
Specifies the database user ID.

PASSWORD
Specifies the database password.

RPT_MAX_ROWS
Specifies the maximum number of rows within a table that a function will return in a report.

SHOWSQL
Specifies whether Net.Data should hide or display the SQL statement being executed.

user_defined_variable
Variables that are passed to Net.Data and provide required information or effect Net.Data behavior. User-defined variables are variables that you define for your application.

VALUE
Specifies the value of the Net.Data variable.

LANGENV
Specifies the target language environment for the SQL statement or stored procedure call. If the language environment is one of the database language environments, the database name must also be specified.

dblangenv
The name of the database language environment:

- DTW_SQL
- DTW_ODBC
- DTW_ORA
- DTW_SYB

SQL
Indicates that the direct request specifies the execution of an in-line SQL statement.

query string
Specifies a string that contains any valid SQL statement that can be executed using dynamic SQL.

FUNC
Indicates that the direct request specifies the execution of a stored procedure.

stored_proc_name
Specifies any valid DB2 stored procedure name.

parm_type
Specifies any valid parameter type for a DB2 stored procedure.

parm_name
Specifies any valid parameter name.

parm_value
Specifies any valid parameter value for a DB2 stored procedure.

IN Specifies that Net.Data should use the parameter to pass input data to the stored procedure.

INOUT
Specifies that Net.Data should use the parameter to both pass input data to the stored procedure and return output data from the language environment.

OUT
Specifies that the language environment should use the parameter to return output data from the stored procedure.

Non-database language environment call
Specifies a direct request to Net.Data that invokes a non-database language environment.

LANGENV
Specifies the target language environment for the execution of the function.

lang_env
Specifies the name of the non-database language environment:

- DTW_PERL
- DTW_REXX
- DTW_SYSTEM

FUNC
Indicates that the direct request specifies the execution of a program.

program_name
Specifies the program containing the function to be executed.

parm_value
Specifies any valid parameter value for the function.

Direct Request Examples

The following examples show the different ways you can invoke Net.Data while using the direct request method.

HTML Links

Example 1: A link that invokes the Perl language environment and calls a Perl script that is in the EXEC path statement of the Net.Data initialization file

```
<A HREF="http://server/cgi-bin/db2www?LANGENV=DTW_PERL&FUNC=my_perl(hi)">
  any text</A>
```

Example 2: A link that invokes the Perl language environment, as in the previous example, but passes a string with URL-encoded values for the double quote and the space characters

```
<A HREF="http://server/cgi-bin/db2www?LANGENV=DTW_PERL&FUNC=my_perl
     (%22Hello+World%22)">any text</A>
```

Tip: You must encode certain characters, such as spaces and double quotes, within URLs. In this example, the double quotes characters and spaces within the parameter value must be encoded as %22 and the + character, respectively. For a list of all the characters that must be encoded within URLs, see the description of DTW_URLESCSEQ function in *Net.Data Reference*.

HTML Forms

Example 1: An HTML form that results in the execution of an SQL query using the SQL language environment, connects to the CELDIAL database, and queries a table

```
<FORM METHOD="POST"
 ACTION="http://server/cgi-bin/db2www/">
<INPUT TYPE=hidden NAME="LANGENV" VALUE="DTW_SQL">
<INPUT TYPE=hidden NAME="DATABASE" VALUE="CELDIAL"
 <INPUT TYPE=hidden NAME="SQL" VALUE="select * from Table1 where col1=$(InputName)">
Enter Customer name:
<INPUT TYPE=text NAME="InputName" VALUE="John">
<INPUT TYPE=SUBMIT>
</FORM>
```

Tip: You can use variable substitutions in direct request invocations by using an HTML form that is created by a Net.Data macro.

URL

Example 1: A URL that results in the execution of an SQL query using the SQL language environment

```
http://server/cgi-bin/db2www?LANGENV=DTW_SQL&DATABASE=CELDIAL
 &SQL=select+*+from+customer
```

Example 2: A URL that invokes the Perl language environment and calls an executable file that is not in the EXEC path statement of the Net.Data initialization file

```
http://server/cgi-bin/db2www?LANGENV=DTW_PERL&FUNC=/u/MYDIR/macros/myexec.pl
```

Tip: To reference a file name with a path that is not specified in the EXEC path configuration statement, provide the fully qualified path along with the file name for the *function_name* value.

Example 3: A URL that invokes the System language environment and calls an external Perl script

```
http://server/cgi-bin/db2www?LANGENV=DTW_SYSTEM&FUNC=perl+/u/MYDIR/macros/myexec.pl
```

Example 4: A URL that invokes the REXX language environment, calls a REXX program, and passes parameters to the program

```
http://server/cgi-bin/db2www?LANGENV=DTW_REXX&FUNC=myexec.cmd(parm1,parm2)
```

Example 5: A URL that calls a stored procedure and passes parameters to the SQL language environment

```
http://server/cgi-bin/db2www?LANGENV=DTW_SQL&FUNC=MY_STORED_PROC
   (IN+CHAR(30)+Salaries)&DATABASE=CELDIAL
```

Chapter 5. Developing Net.Data Macros

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 built-in to Net.Data or defined in the macro file
- Format the processing output in HTML and return it to the Web browser

The Net.Data macro contains two parts: the declaration part and the HTML part, as shown in Figure 19.

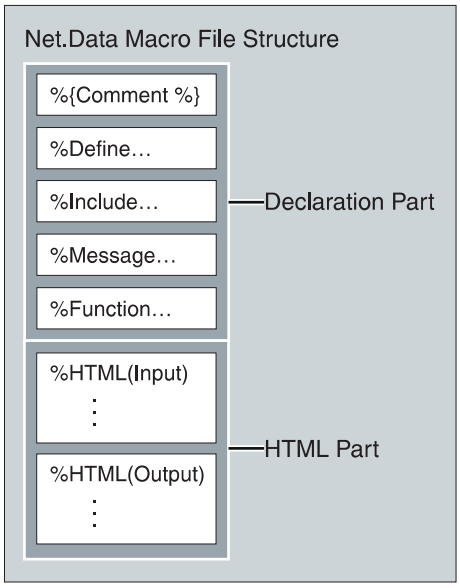


Figure 19. Macro File Structure

- The *declaration part* contains the definitions of variables and functions in the macro file.
- The *HTML part* contains HTML blocks made up of HTML statements that specify the layout of the Web page.

You can use these parts multiple times and in any order. See *Net.Data Reference* for syntax of the macro file parts and constructs.

Authorization Tip: Ensure that the Web server has access rights to this file. See "Specifying Access Rights to Net.Data Files" on page 40 for more information.

This chapter examines the different blocks that make up a Net.Data macro file and methods you can use for writing the macro file.

- “Using Large Objects” on page 91

Anatomy of a Net.Data Macro File

The macro file consists of two parts:

- The declaration part, that contains definitions used in the HTML part. The declaration part uses two major optional blocks:
  - DEFINE block
  - FUNCTION block

  The declaration part can also contain other language constructs and statements, such as EXEC statements, IF blocks, INCLUDE statements, and MESSAGE blocks. For more information about the language constructs, see the chapter about language constructs in *Net.Data Reference*.

  Authorization Tip: Ensure that the Web server has access rights to files referenced by the EXEC and INCLUDE statements. See “Specifying Access Rights to Net.Data Files” on page 40 for more information.
- The HTML part defines the layout of the Web page, references variables, and calls functions using HTML blocks that are used as entry and exit points from the macro. When you invoke Net.Data, you specify an HTML block name as an entry point for processing the macro file. The HTML blocks are described in “HTML Blocks” on page 60.

In this section, a simple Net.Data macro illustrates the elements of the macro language. This example macro presents a form that prompts for information to pass to a REXX program. The macro passes this information to an external REXX program called `OMPSAMP.CMD`, which echoes the data that the user enters. The results are then displayed on a second HTML page.

First, look at the entire macro, and then each block in detail:

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

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

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

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

<FORM  METHOD="post" ACTION="output">
Type some data to pass to a REXX program:
```

```
<INPUT NAME="input_data" TYPE="text" SIZE="30">
<p>
<INPUT TYPE="submit" VALUE="Enter">

</form>

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

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

The sample macro consists of four major blocks: the DEFINE, the FUNCTION, and the two HTML blocks. You can have multiple DEFINE, FUNCTION, and HTML blocks in one Net.Data macro.

The two HTML blocks contain familiar HTML tags, which make writing Web macros easy. If you are familiar with HTML, building a macro simply involves adding macro statements to be processed dynamically at the server and SQL statements to send to the database.

Although the macro looks similar to an HTML document, the Web server accesses it through Net.Data using CGI or a Web server API. Net.Data requires two parameters: the name of the macro to process, and the HTML block in that macro to display.

When the macro file is invoked, Net.Data processes it from the beginning. The following sections look at what happens as Net.Data processes the file.

The DEFINE Block

The DEFINE block contains the DEFINE language construct and variable definitions used later in the HTML blocks. The following example shows a DEFINE block with one variable definition:

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

The first line is a comment. A comment is any text inside `%{` and `%}`. Comments can be anywhere in the macro file. The next statement starts a DEFINE block. You can define multiple variables in one define block. In this example, only one variable, `page_title`, is defined. After it is defined, this variable can be referenced anywhere in the macro using the syntax, `$(page_title)`. Using variables makes it easy to make global changes to your macro later. The last line of this block, `%}`, identifies the end of the DEFINE block.

The FUNCTION Block

The FUNCTION block contains declarations for functions invoked by the HTML blocks. Functions are processed by language environments and can execute programs, SQL queries, or stored procedures.

The following example shows two FUNCTION blocks that define a function call to an external REXX program and a function call to a function contained within the macro file.

```
%{ ********************** FUNCTION Block ***********************************%}
%FUNCTION(DTW_REXX) rexx1 (IN input) returns(result) { <-- This function accepts
                                                          one parameter and returns a
                                                          result which is substituted
                                                          for the associated function
                                                          call
     %EXEC{ompsamp.cmd %}  <-- The function executes an external REXX program
                               called "ompsamp.cmd"
%}

%FUNCTION(DTW_REXX) today () RETURNS(result) {
      result = date()  <-- The single source statement for this function is
                           contained inline.
%}
```

The first function block, `rexx1`, is a REXX function declaration that in turn, runs an external REXX program called `ompsamp.cmd`. One input variable, `input`, is accepted by this function and automatically passed to the external REXX command. The REXX command also returns one variable called `result`. The contents of the `result` variable in the REXX command replaces the invoking `@rexx1()` function call contained in the Output block. The variables `input` and `result` are directly accessible by the REXX program, as shown in the source code for `ompsamp.cmd`:

```
/* REXX */
result = 'The REXX program received "'input'" from the macro.'
```

The code in this function echoes the data that was passed to it. You can format the resulting text any way you want by enclosing the requesting `@rexx1()` function call in normal HTML style tags (like <b> or). Rather than using the `result` variable, the REXX program could have written HTML tags to standard output using REXX SAY statements.

The second block, `today`, also refers to a REXX program. However, the entire REXX program (one whole line) in this case is contained in the function declaration itself. An external program is not needed. Inline programs are allowed for both REXX and Perl functions because they are interpreted languages that can be parsed and executed dynamically. Inline programs have the advantage of simplicity by not requiring a separate program file to manage. The first REXX function could also have been handled inline.

HTML Blocks

HTML blocks defines the layout of the Web page, references variables, and calls functions using HTML blocks that are used as entry and exit points from the macro. An HTML is always specified in the Net.Data invocation request and every macro must have at least one HTML block.

The first HTML block in the example macro file is named `INPUT`. The INPUT block contains the HTML for a simple form with one input field.

```
%{ **********************      HTML Block: Input    ************************%}
%HTML (INPUT) {               <--- Identifies the name of this HTML block.
<html>
<head>
<title>$(page_title)</title>  <--- Note the variable substitution.
</head><body>
<h1>Input Form</h1>
Today is @today()             <--- This line contains a call to a function.

<FORM METHOD="post" ACTION="output">  <--- When this form is submitted,
                                           the "output" HTML block is called.
Type some data to pass to a REXX program:
<INPUT NAME="input_data"      <--- "input_data" is defined when the form
TYPE="text" SIZE="30">             is submitted and can be referenced elsewhere in
                                   this macro. It is initialized to whatever the
                                   user types into the input field.
<p>
<INPUT TYPE="submit" VALUE="Enter">

<hr>
<p>
[
<a href="/">Home page</a>]
 </body><html>
%}                            <--- Closes the HTML block.
```

The entire block is surrounded by the HTML block identifier, `%HTML (INPUT) {...%}`. `INPUT` identifies the name of this block. You can give it any name. The HTML `<title>` tag contains an example of variable substitution. The value of the variable `page_title` is substituted into the title of the form.

This block also has a function call. The expression `@today()` is a call to the function `today`. This function is defined in the FUNCTION block that is described later. Net.Data inserts the result of the `today` function, the current date, into the HTML text in the same location that the `@today()` expression is located.

The ACTION parameter of the FORM statement provides an example of navigation between HTML blocks or between macros. Referencing the name of another block in an ACTION parameter accesses that block when the form is submitted. Any input data from an HTML form is passed to the block as implicit variables. This is true of the single input field defined on this form. When the form is submitted, data typed in this field is passed to the HTML output block in the variable *input_data*.

You can access HTML blocks in other macro files with a relative reference if the macro files are on the same Web server. For example, the ACTION parameter `ACTION="../othermacro.d2w/main"` accesses the HTML block called main in the macro file `othermacro.d2w`. Again, any data typed in the form is passed to this macro in the variable *input_data*.

When you invoke Net.Data, you pass the variable as part of the URL. For example:

```
<a href="/cgi-bin/db2www/othermacro.d2w/main?input_data=value">Next macro</a>
```

You don't need to define environment variables to receive input data, as you would with most CGI programs. Net.Data handles environment variables for you. You only need to reference the variable names.

The next HTML block in the example is the `OUTPUT` block. It contains the HTML tagging and Net.Data macro statements that define the output processed from the INPUT block request.

```
%{ **********************    HTML Block: Output     ************************%}
%HTML (OUTPUT) {
<html>
<head>
<title>$(page_title)</title>  <--- More substitution.

</head><body>
<h1>Output Page</h1>
<p>@rexx1(input_data)  <--- This line contains a call to function rexx1
                           passing the argument "input_data".
<p>
<hr>
<p>
[
<a href="/">Home page</a> |
<a href="input">Previous page</a>]
%}
```

Like the INPUT block, this block is standard HTML with Net.Data macro statements to substitute variables and a function call. Again the `page_title` variable is substituted into the title statement. And, as before, this block contains a function call. In this case, it calls the function `rexx1` and passes to it the contents of the variable `input_data`, which it received from the form defined in the Input block. You can pass any number of variables to and from a function. The function definition determines the number and types of variables that are passed.

Net.Data Macro Variables

Net.Data lets you define and reference variables in a Net.Data macro. In addition, you can pass these variables from the macro to language environments and back.

Net.Data variables can be defined depending on the type of variable and whether is has a predefined value. These variables can be categorized into the following types, based on how they are defined:

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

An identifier, which is a variable or a function call, becomes *visible*, meaning that it can be referenced when it is declared or instantiated. The region where an identifier is visible is called its *scope*. The five types of scope are:

- Global

  An identifier has global scope if you can reference it anywhere within a macro file. Identifiers that have global scope are:
  - Net.Data built-in functions
  - Form data

  - URL data
  - Variables instantiated from within an HTML block
- Macro file

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

  An identifier has function block scope if it is declared in the parameter list of the function definition. If an identifier with the same name already exists outside the function definition, Net.Data uses the identifier from the function parameter list within the function block. Net.Data does not use or modify the identifier or its value from outside the function block.

  An identifier does not have function block scope if it has been declared or initialized outside of the function and is not declared in the function parameter list. When the identifier is used inside the function block, it keeps the value it has been assigned to prior to the function call. When updated inside the function block, the identifier keeps the new value after the function call.

  An identifier has function block scope when it is not declared in the parameter list and not declared or initialized prior to the function call. It cannot be referenced outside of the function block.
- REPORT block

  An identifier has report block scope if it can be referenced only from within a REPORT block (for example, table column names N1, N2, ..., N*n*). Only those variables that Net.Data implicitly defines as part of its table processing can have a report block scope. Any other variables that are instantiated have function block scope.
- ROW block

  An identifier has row block scope if it can only be referenced from within a ROW block (for example, table value names V1, V2, ..., Vn). Only those variables that Net.Data implicitly defines as part of its table processing can have a row block scope. Any other variables that are instantiated have function block scope.

When an identifier is referenced, it is replaced with the value of the identifier. If a reference to a variable has no value associated with it, or if a function call does not have a return value, the reference is replaced by an empty string.

The following sections describe how to define and reference variables, and also describe different variable types and how to use them.

Defining Variables

The three ways to define variables in a Net.Data macro are:

- **DEFINE statement or block**

  The simplest way to define a variable for use in a Net.Data macro is to use the DEFINE statement. This syntax is specific to Net.Data:

```
%DEFINE variable_name="variable value"

%DEFINE variable_name={ variable value on multiple
                        lines of text  %}
```

The *variable_name* is the name you give the variable. Variable names must begin with a letter or underscore and can contain any alphanumeric characters or an underscore. All variable names are case-sensitive, except N_*columnName* and V_*columnName*, which are table variables.

To include quotes in a string, use two quotes consecutively. Two consecutive quotes alone is equal to a null string. For example:

```
%DEFINE HI="say ""hello"""
```

The variable `HI` displays `say "hello".`

```
%DEFINE reply="hello"
```

The variable `reply` displays `hello`,

```
%DEFINE empty=""
```

The variable `empty` is null.

To define several variables with one DEFINE statement, use a DEFINE block:

```
%DEFINE{
         variable1="value1"
         variable2="value2"
         variable3="value3"
         variable4="value4"
%}
```

- **HTML form SELECT and INPUT tags**

  You can use the SELECT and INPUT tags used for an HTML form. The following example uses standard HTML form tags to define a variable:

  ```
  <INPUT NAME="variable_name" TYPE=...>
  ```

  or

  ```
  <SELECT NAME="variable_name">
  ```

  The *variable_name* is the name you give the variable, and the value of the variable is determined from the input received in the form. See “HTML Forms” on page 50 for an example of how this type of variable definition is used in a Net.Data macro.

  A variable value received from an INPUT or SELECT tag overrides a variable value set by a DEFINE statement in a Net.Data macro.

- **URL data**

  You can call Net.Data macros as URL requests and include variables, such as a user ID, in the URL to send to Net.Data. For example:

  ```
  http://www.ibm.com/cgi-bin/db2www/stdqry1.d2w/input?field=custno
  ```

  In the above example, the variable name, `field`, and the variable value, `custno`, specify additional data that Net.Data receives from the input statement. Net.Data receives and processes the data as it would form data.

Referencing Variables

You can reference a previously defined variable to return its value.

To reference a variable in Net.Data macros, specify the variable name inside $(and). For example:

```
$(variableName)
$(homeURL)
```

When Net.Data finds a variable reference, it substitutes the variable reference with its value.

To use variables as part of your HTML, reference them in your HTML blocks. For example, if you have defined the variable homeURL:

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

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

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

You can reference variables in any part of a Net.Data macro. If the variable has not yet been defined at the time it is referenced, Net.Data returns an empty string. Net.Data does not define the variable.

Restriction: Circular references (or cycles) are not allowed. For example, the DEFINE statements below result in an error when the variable is referenced and the final values are evaluated:

```
%DEFINE a="$(b)"
%DEFINE b="$(a)"
```

Variable Types

You can use the following types of variable references in your macro files.

- "Conditional Variables"
- "Environment Variables" on page 66
- "Executable Variables" on page 66
- "Hidden Variables" on page 68
- "List Variables" on page 68
- "Table Variables" on page 69
- "Miscellaneous Variables" on page 70
- "Table Processing Variables" on page 70
- "Report Variables" on page 71
- "Language Environment Variables" on page 72

If you assign strings to variables that are defined a certain way by Net.Data, such as ENVVAR, LIST, condition list variables, the variable no longer behaves in the defined way. In other words, the variable becomes a simple variable, containing a string.

Conditional Variables

Conditional variables let you define a conditional value for a variable by using a method similar to an IF, THEN construct. When defining the conditional variable, you can specify two possible variable values. If the first variable you reference

exists, the conditional variable gets the first value; otherwise the conditional variable gets the second value. The syntax for a conditional variable is:

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

If `varB` is defined, `varA="value_1"`, otherwise `varA="value_2"`. This is equivalent to using an IF block, as in the following example:

```
%IF $(varB)
    varA = "value_1"
%ELSE
    varA = "value_2"
%ENDIF
```

See “List Variables” on page 68 for an example of using conditional variables with list variables.

Environment Variables

You can reference Net.Data environment variables that exist in the process under which Net.Data is running.

The syntax for defining environment variables is:

```
%define var=%ENVVAR
```

Where *var* is the name of the variable being defined.

For example, the variable SERVER_NAME can be defined as environment variable:

```
%define SERVER_NAME=%ENVVAR
```

And then referenced:

```
The server is $(SERVER_NAME)
```

The output looks like this:

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

See *Net.Data Reference* for more information about the ENVVAR statement.

Executable Variables

You can invoke other programs from a variable reference using executable variables.

Define executable variables in a Net.Data macro using the EXEC language construct in the DEFINE block. For more information about the EXEC language element, see the language constructs chapter in the *Net.Data Reference*. In the following example, the variable `runit` is defined to execute the executable program `testProg`:

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

`runit` becomes an executable variable.

Net.Data runs the executable program when it encounters a valid variable reference in a Net.Data macro. For example, the program `testProg` is executed when a valid 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. The following example demonstrates this method. The variable `date` is defined as an executable variable and `dateRpt` is then defined as a variable reference, which contains the executable variable.

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

Wherever `$(dateRpt)` appears in the Net.Data macro, Net.Data searches for the executable program `date`, and when it locates it, returns:

```
Today is Tue 11-07-1999
```

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

1. It searches the directories specified by the EXEC_PATH in the Net.Data initialization file. See “EXEC_PATH” on page 14 for details.
2. If Net.Data does not locate the program, the system searches the directories defined by the system PATH environment variable or the library list. If it locates the executable program, Net.Data runs the program.

Restriction: Do not set an executable variable to the value of the output of the executable program it calls. In the previous example, the value of the variable `date` is null. If you use this variable 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. In this example, the values of distance and time are passed to the program *calcMPH*.

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

This next example returns the system date as part of the HTML report:

```
%DEFINE database="celdial"
 %DEFINE tstamp=%exec "date"

%FUNCTION(DTW_SQL) myQuery() {
SELECT CUSTNO, CUSTNAME from dist1.customer
%REPORT{
%ROW{
<A HREF="/cgi-bin/db2www/exmp.d2w/report?value1=$(V1)&value2=$(V2)">
$(V1) $(V2) </A> <BR>
%}
%}
%}

%HTML(report){
<H1>Report made: $(tstamp) </H1>
@myQuery()
%}
```

Each report displays the date for easy tracking. This example also puts the customer number and name in a link for another Net.Data macro. Clicking on any customer in the report calls the `exmp.d2w` Net.Data macro, passing the customer number and name to the Net.Data macro.

Hidden Variables

You can use hidden variables to conceal the actual name of a variable from application users who view your HTML source with their Web browser. To define a hidden variable:

1. Define a variable for each string you want to hide, after the variable's last reference in the HTML block. Variables are always defined with the DEFINE language construct after they are used in the HTML block, as in the following example. The $$(*variable*) variables are referenced and then defined.
2. In the HTML block where the variables are referenced, use double dollar signs instead of a single dollar sign to reference the variables. For example, `$$(X)` instead of `$(X)`.

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

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

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

.
.
.
```

When a Web browser displays the HTML form, `$$(name)` and `$$(addr)` are replaced with `$(name)` and `$(addr)` respectively, so the actual table and column names never appear on the HTML form. Application users cannot tell that the true variable names are hidden. When the user 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

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 syntax for a list variable is:

```
%LIST " value_separator " variable_name
```

Recommendation: The blanks are significant. Insert a space before and after the value separator for most cases. Most queries use Boolean or mathematical operators (for example, AND, OR, or >) for the value separator. The following example illustrates the use of conditional, hidden, and list variables:

```
%HTML(INPUT){
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/example2.d2w/report">
<H2>Select one or more cities:</H2>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$(cond1)">Sao Paola<BR>
```

```
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$(cond2)">Seattle<BR>
<INPUT TYPE="checkbox" NAME="conditions" VALUE="$$(cond3)">Shanghai<BR>
<INPUT TYPE="submit" VALUE="Submit Query">
</FORM>
%}

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

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

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

In the HTML form, if no boxes are checked, `conditions` is null, so `whereClause` is also null in the query. Otherwise, `whereClause` has the selected values separated by 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'
```

This example shows that `Seattle` is selected, which results in this SQL query:

```
SELECT name, city FROM citylist
WHERE cond1='Seattle'
```

Table Variables

The table variable defines a collection of related data. It contains an array of identical records, or rows, and an array of column names describing the fields in each row. A table is defined in the Net.Data macro as in the following statement :

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

The number following TABLE is the limit on the number of rows this table can contain. To specify a table with no limit on the number of rows, use the default or specify ALL, as shown in these examples:

```
%DEFINE myTable2=%TABLE
%DEFINE myTable3=%TABLE(ALL)
```

When you define a table, it has zero rows and zero columns, and no storage is allocated. The only way you can populate a table with values is by passing it as an OUT or INOUT parameter to a function. The DTW_SQL language environment automatically puts the results of a SELECT statement into a table.

For all other language environments, such as DTW_REXX or DTW_PERL, the language environment does not automatically set table values. Instead, the individual elements of the table are made available to the native language interpreter as output parameters, and must then be set by the script or program being executed. See the language environment descriptions in the *Language Environment Reference* for details.

You can pass a table between functions by referring to the table variable name. The individual elements of the table can be referred to in a REPORT block of a function. See “Table Processing Variables” for details. Table variables are usually populated with values in an SQL function, and then used as input to a report, either in the SQL function or in another function after being passed to that function as a parameter. You can pass table variables as IN, OUT, or INOUT parameters to any non-SQL function. Tables can be passed to SQL functions only as OUT parameters.

The column names and field values in a table are addressed as array elements with an origin of 1, rather than the standard C and C++ language convention of starting arrays at 0.

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
- Obtain information about the result set of a database query
- Determine information about file locations and dates

Miscellaneous variables can either have a predefined value that Net.Data determines or have values that you set. For example, Net.Data determines the DTW_CURRENT_FILENAME variable value based on the current file that it is processing, whereas you can specify whether Net.Data removes extra white space caused by tabulators and new-line characters.

Pre-defined variables are used as variable references within the macro file and provide information about the current status of files, dates, or the status of a function call. For example, to retrieve the name of the current file, you could use:

```
<p>This file is <i>$(DTW_CURRENT_FILENAME)</i>.</P>
```

Modifiable variable values are generally set using a DEFINE statement or with the @DTW_ASSIGN() function and let you affect how Net.Data processes the macro file. For example, to specify whether white space is removed, you could use the following DEFINE statement:

```
%DEFINE DTW_REMOVE_WS="YES"
```

See the variables chapter in *Net.Data Reference* for a list of valid miscellaneous variables with syntax and examples.

Table Processing Variables

Net.Data defines table processing variables for use in the REPORT and ROW blocks. Use these variables to reference values from SQL queries and function calls.

Table processing variables have a predefined value that Net.Data determines and allow you to reference values from the result sets of SQL queries or function calls by column, row, or field that is being processed. You can also access information about the number of rows being processed or a list of all the column names.

For example, as Net.Data processes a result set from an SQL query, it assigns the value of the variable Nn for each current column name, such that N1 is assigned to the first column, N2 is assigned to the second column, and so on. You can reference the current column name for your HTML output.

Use table processing variables as variable references within the macro file. For example, to retrieve the name of the current column being processed, you could use:

```
<p>Column 1 is <i>$(N1)</i>.</P>
```

Table processing variables also provide information about the results of a query. You can reference the variable TOTAL_ROWS in the macro to show how many rows are returned from an SQL query, as in the following example:

```
Names found: $(TOTAL_ROWS)
```

Some of the table processing variables are affected by other variables or built-in functions. For example, TOTAL_ROWS requires that the DTW_SET_TOTAL_ROWS SQL language environment variable be activated so that Net.Data assigns the value of TOTAL_ROWS when processing the results from a SQL query or function call as in the following example:

```
%DEFINE DTW_SET_TOTAL_ROWS="YES"
...

Names found: $(TOTAL_ROWS)
```

See the variables chapter in *Net.Data Reference* for a list of valid table processing variables with syntax and examples.

Report Variables

Net.Data displays HTML output generated from the macro file in a default report format. The default report format displays in a table format using <PRE> </PRE> tags. You can override the default report by defining a REPORT block with instructions for displaying the output or by using one of the report variables to prevent the default report from being generated.

Report variables help you customize how your HTML output is displayed and used with default reports and with Net.Data tables. You must define these variables before using them with a DEFINE statement or with the @DTW_ASSIGN() function.

The report variables specify spacing, override default report formats, specify HTML table output versus default table output, and other display features. For example, you can use the ALIGN variable to control leading and trailing spaces for table processing variables. The following example uses the ALIGN variable to separate by a space each column name in a list that is returned by a query.

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

The START_ROW_NUM report variable lets you determine at which row to begin displaying the results of a query. For example, the following variable value specifies that Net.Data will begin displaying the results of a query at the third row.

```
%DEFINE START_ROW_NUM = "3"
```

You can also determine whether Net.Data uses HTML tags for default formatting. With DTW_HTML_TABLE set to YES, an HTML table is generated rather than a text-formatted table.

```
%DEFINE DTW_HTML_TABLE="YES"

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

See the variables chapter in *Net.Data Reference* for a list of valid report variables with syntax and examples.

Language Environment Variables

These variables are used with language environments and affect how the language environment processes a request. You must define these variables before referencing them, using a DEFINE statement or the @DTW_ASSIGN() function. Set or reference language environment variables in the Net.Data macro blocks that are appropriate.

With these variables, you can perform tasks like establishing connections with databases, supply alternate text for Java applets, and enabling NLS support.

For example, you can use the SQL_STATE variable to access or display the SQL state value returned from the database.

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

This next example shows how to define which database is to accessed.

```
%DEFINE DATABASE="CELDIAL"
```

See the variables chapter in *Net.Data Reference* for a list of valid language environment variables with syntax and examples.

Net.Data Functions

Net.Data provides many functions you will find useful in your Web applications. It is easy to write your own functions, too. Figure 20 on page 73 shows how the Net.Data functions and the macro file interact.

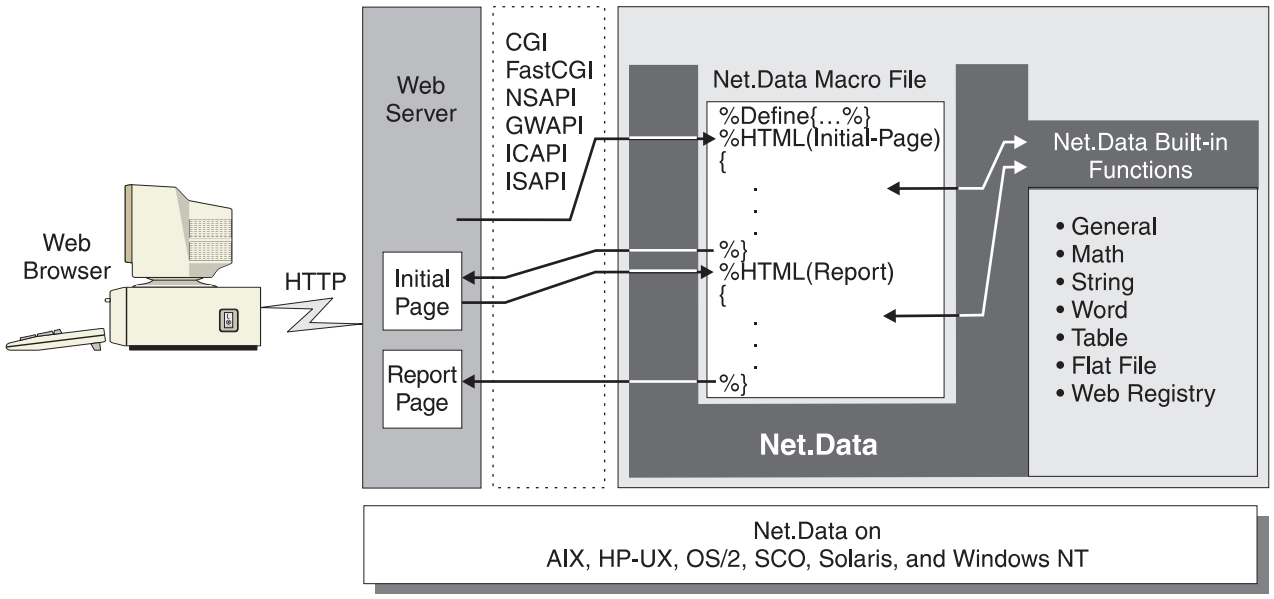


Figure 20. Net.Data Built-in Functions

Defining Functions

You can define your own functions or use Net.Data's built-in functions. To define your own functions to the macro file (called user-defined functions), use a FUNCTION block or MACRO_FUNCTION block.

FUNCTION block
Defines a subroutine that is invoked from a Net.Data macro and that is either processed by language environment or calls an external program.

MACRO_FUNCTION block
Defines a subroutine that is invoked from a Net.Data macro and must be processed by Net.Data, rather than another language environment. The statements in the block must be Net.Data macro language source statements.

The MACRO_FUNCTION block is an alternative to the FUNCTION block that can improve performance. The MACRO_FUNCTION block is processed only by Net.Data and does not call a language environment. Details about these two constructs are in *Net.Data Reference*.

The syntax is shown below:

FUNCTION block:

```
%FUNCTION(type) function-name(usage parameter, ...) [RETURNS(return-var)] {
  executable-statements
  report-block
  message-block
%}
```

MACRO_ FUNCTION block:

```
%MACRO_FUNCTION function-name(usage parameter, ...)  {
  executable-statements
    %}
```

Where:

type

Identifies a language environment that is configured in the initialization file. The language environment invokes a specific language processor (which processes the executable statements) and provides a standard interface between Net.Data and the language processor.

Several default language environments are provided with Net.Data.

function-name

Specifies the name of the FUNCTION or MACRO_FUNCTION block. Execute the FUNCTION block or MACRO_FUNCTION block with a function call elsewhere in the macro file. The function call references the *function-name*, preceded by an at (@) sign. See "Calling Functions" on page 76 for details.

You can define multiple FUNCTION or MACRO_FUNCTION blocks with the same name so that they are processed at the same time. Each of the blocks must all have identical parameter lists. When Net.Data calls the function, all FUNCTION blocks with the same name or MACRO_FUNCTION blocks with the same name are executed in the order they are defined in the Net.Data macro.

usage Specifies whether a parameter is an input (IN) parameter, an output (OUT) parameter, or both types (INOUT). This designation indicates whether the parameter is passed into or received back from the FUNCTION block, MACRO_FUNCTION block, or both. The usage type applies to all of the subsequent parameters in the parameter list until changed by another usage type. The default type is IN.

parameter

The name of a variable with local scope that is replaced with the value of a corresponding argument specified on a function call. Parameter references, for example $(*parm1*), in the executable statements or REPORT block are replaced with the actual value of the parameter. In addition, parameters are passed to the language environment and are accessible to the executable statements using the natural syntax of that language or as environment variables. Parameter variable references are not valid outside the FUNCTION block or MACRO_FUNCTION block.

return-var

Specify this parameter after the RETURNS keyword to identify a special OUT parameter. The value of the return variable is assigned to the function call and replaces the function call in the Net.Data macro processing. If you do not specify the RETURNS clause, the value of the function call is:

- NULL if the return code from the call to the language environment is zero
- The value of the return code, when the return code is non-zero.

executable-statements

The set of language statements that is passed to the specified language environment for processing after the variables are substituted and the functions are processed. *executable-statements* can contain Net.Data variable references and Net.Data function calls. Prior to passing the executable statements to the language environment, Net.Data replaces these variable references or function calls with the actual value. *Executable-statements* includes those executable statements that are allowed in an HTML block.

For FUNCTION blocks, Net.Data replaces all variable references with the variable values, executes all function calls, and replaces the function calls with their resulting values before the executable statements are passed to the language environment. Each language environment processes the statements differently. For more information about specifying executable statements or calling executable programs, see “Executable Variables” on page 66.

For MACRO_FUNCTION blocks, the executable statements are a combination of HTML statements and Net.Data macro language constructs. In this case, no language environment is involved because Net.Data acts as the language processor and evaluates and runs the executable statements.

report-block
Defines the REPORT block for handling the output of the FUNCTION block. See “Report Blocks” on page 86.

message-block
Defines the MESSAGE block, which handles any messages returned by the FUNCTION block. See “Message Blocks” on page 83.

Define functions at the outermost Net.Data macro layer and before they are called in the Net.Data macro.

Using Special Characters in Language Statements

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 or Perl), they can be misinterpreted as Net.Data language constructs, causing errors or unpredictable results in a macro.

For example, a Perl function might use the COMMENT block delimiter characters, %{. When the macro is run, the %{ characters are interpreted as the beginning of a COMMENT block. Net.Data then looks for the end of the COMMENT block, which it thinks it finds when it reads the end of the function block. Net.Data then proceeds to look for the end of the function block, and when it can’t be found, issues an error.

Use one of the following methods to use COMMENT block delimiter characters, or any other 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.

For example, the following Perl function contains characters representing a COMMENT block delimiter, %{, as part of its Perl language statements:

```
%function(DTW_PERL) func() {
  ...
  for $num_words (sort bynumber keys %{ $Rtitles{$num} }) {
     &make_links($Rtitles{$num}{$num_words});
  }
  ...
 %}
```

To ensure that Net.Data interprets the %{ characters as Perl source code rather than as a Net.Data COMMENT block delimiter, rewrite the function in either of the following ways:

- Use the %EXEC statement:

```
%function(DTW_PERL) func() {
  %EXEC{ func.prl %}
%}
```

- Use a variable reference to specify the %{ characters:

```
%define percent_openbrace = "%{"

%function(DTW_PERL) func() {
 ...
 for $num_words (sort bynumber keys $(percent_openbrace) $Rtitles{$num} }) {
  &make_links($Rtitles{$num}{$num_words});
 }
 ...
%}
```

Calling Functions

Call a function from a Net.Data macro using the at (@) character followed by a FUNCTION block name or a MACRO_FUNCTION block name:

```
@function_name([ argument,... ])
```

function_name
: This is the name of the FUNCTION block

  or MACRO_FUNCTION block to invoke. The function must already be defined in the Net.Data macro, unless this is a built-in function.

argument
: This is the name of a defined variable, a literal character string, a variable reference, or a function call. Arguments on a function call are matched up with the parameters on a FUNCTION block or MACRO_FUNCTION block, and each parameter is assigned the value of its corresponding argument while the FUNCTION block or MACRO_FUNCTION block is being processed. The arguments must be the same number and type as the corresponding parameters.

Net.Data processes the FUNCTION block, MACRO_FUNCTION block, or built-in functions associated with a function call in the following order:

1. Net.Data processes the variable references and function calls in the executable-statements section of the FUNCTION block. Net.Data replaces all variable references with the current values of the variables, and executes and replaces all function calls with the return value of the function call. The variable references and function calls are processed in the order in which they are specified. Net.Data does not process built-in functions or MACRO_FUNCTION blocks during this step.
2. The native language processor processes the executable-statements section. For FUNCTION blocks, the processor corresponds to the language environment specified on the FUNCTION block, such as SQL, REXX, or Perl. For MACRO_FUNCTION blocks, Net.Data acts as the language processor and executes the executable statements. Built-in functions do not have executable statements. Net.Data processes the built-in functions by function name.

   Net.Data passes the function's parameters to the native language processor. Net.Data passes values to the native language processor only for IN and INOUT parameters and accepts returned values from the native language processor only for OUT and INOUT parameters,

3. Net.Data sets the implicit RETURN_CODE and DTW_DEFAULT_MESSAGE variables based on the return code and the returned message from the language processor. Net.Data does not set these variables for MACRO_FUNCTION blocks.
4. If the FUNCTION block contains a REPORT block, or specifies a default report to be generated, Net.Data processes the report section using the new values of any referenced output parameters. Net.Data does not generate reports for built-in functions.
5. If the FUNCTION block contains a local MESSAGE block, Net.Data processes the MESSAGE block. Net.Data processes the global MESSAGE block when one of the following conditions occurs:
   - A global MESSAGE block is specified and the return code is not handled by a local MESSAGE block.
   - A built-in function is called.

   Net.Data does not process MESSAGE blocks or MACRO_FUNCTION blocks.
6. Net.Data replaces the function call with the returned value of the function. For FUNCTION blocks, this value is one of the following:

   RETURNS parameter value
   Substituted for the FUNCTION block with a RETURNS keyword.

   Empty string ("")
   Substituted For the FUNCTION block without a RETURNS keyword when RETURN_CODE is zero.

   RETURN_CODE
   Substituted for FUNCTION block without a RETURNS keyword when RETURN_CODE is *not* zero.

   For MACRO_FUNCTION blocks, the results of processing the executable-statements section replaces the function call.

   For built-in functions, the value depends on the format of the built-in function.

Calling Stored Procedures

A stored procedure is a compiled program, stored at the DB2 local or remote server, that can execute SQL statements. In Net.Data, stored procedures are called from Net.Data functions using the CALL SQL statement. Stored procedure parameters are passed in from the Net.Data function parameters list. You can use stored procedures to improve performance and integrity by keeping compiled SQL statements with the database server.

This section describes following topics:

Stored Procedure Syntax

The syntax of the stored procedure uses the FUNCTION statement, the CALL statement, and optionally a REPORT block.

```
%function (dtw_sql) function_name ([IN datatype arg1, INOUT datatype arg2,
    OUT tablename...])
  CALL stored_procedure [(resultsetname....)]
[%REPORT(resultsetname...)]
```

Where:

function_name
Is the name of the Net.Data function that initiates the call of the stored procedure

stored_procedure
Is the name of the stored procedure

datatype
Is one of the database data types supported by Net.Data as shown in Table 5. The data types specified in the parameter list must match the data types in the stored procedure. See your database documentation for more information about these data types.

tablename
Is the name of a Net.Data table in which the result set is to be stored (used only when the result set is to be stored in a Net.Data table). If specified, this parameter name must match the associated parameter name for `resultsetname`.

resultsetname
A name that associates a result set returned from the stored procedure with a report block. This parameter name must match the associated parameter name for `tablename` if specified.

Table 5. Stored Procedures Data Types

BIGINT	FLOAT	SMALLINT
CHAR	INTEGER	TIME
DATE	GRAPHIC	TIMESTAMP
DECIMAL	LONGVARCHAR	VARCHAR
DOUBLE	LONGVARGRAPHIC	VARGRAPHIC
DOUBLEPRECISION		

Calling a Stored Procedure

To call a stored procedure:

1. Define a function that initiates a call to the stored procedure.

   ```
   %function (dtw_sql) function_name()
   ```

2. Optionally, specify any IN, INOUT, or OUT parameters for the stored procedure, including a table name for storing a result set in a Net.Data table (you only need to specify a Net.Data table if you want the result set stored in the Net.Data table).

   ```
   %function (dtw_sql) function_name (IN datatype
   arg1, INOUT datatype arg2, OUT tablename...)
   ```

3. Use the CALL statement to identify the stored procedure name.

   ```
    CALL stored_procedure
   ```

4. If the stored procedure is going to generate one result set, optionally specify a REPORT block to define how Net.Data displays the result set.

   ```
   %REPORT
   ```

   Example:

```
%function (dtw_sql) mystoredproc (IN CHAR(30) arg1)  {
     CALL myproc
 %report {
  ...
  %row {  ....   %}
  ...
 %}
%}
```

5. If the stored procedure is going to generate more than one result set:
   - Specify one or more result set names on the CALL statement.

     CALL *stored_procedure* (*resultsetname1*, *resultsetname2*, ...)
   - Optionally specify one or more REPORT blocks to define how Net.Data displays the result sets.

     %REPORT(*resultsetname1*)

     Example:

     ```
     %function (dtw_sql) mystoredproc (IN CHAR(30) arg1, OUT table1) {
        CALL myproc (table1, table2)
      %report (table2) {
           ...
           %row {  ....   %}
           ...
      %}
     %}
     ```

Passing Parameters

You can pass parameters to a stored procedure and you can have the stored procedure update the parameter values so that the new value is passed back to the Net.Data macro. You pass a parameter in to the stored procedure by specifying the parameter name with IN keyword. If the stored procedure is going to update the parameter, you must pass the parameter for the returned value with the INOUT or OUT keywords. The data type that is specified for a parameter must match what the stored procedure is expecting.

Example 1: Passing a parameter value to the stored procedure

```
%function (dtw_sql) mystoredproc (IN CHAR(30) valuein)  {
    CALL myproc
...
```

Example 2: Returning a value from a stored procedure

```
%function (dtw_sql) mystoredproc (OUT VARCHAR(9) retvalue)  {
  CALL myproc
...
```

Processing Result Sets

You can return one or more result sets from a stored procedure. The result sets can be stored in Net.Data tables for further processing within your macro or displayed using a REPORT block. You must associate a name with each result set generated by the stored procedure. This is done by specifying parameters on the CALL SQL statement. The name you specify for a result set can then be associated with a REPORT block or a Net.Data table, enabling you to determine how each result set is processed by Net.Data. You can:

- Have the result displayed in Net.Data's default report style by not defining a report block for the result set.

- Associate a result set to a REPORT block to have Net.Data display the result set in a report. You can then use Net.Data variables, HTML, or other functions to specify how the report data is displayed at the browser.
- Store the result sets in Net.Data tables when you want Net.Data to use the data later within the macro file. For example, you can pass the Net.Data table to another function so that it can use the data for calculations and display the results based on those calculations.

If a stored procedure generates multiple result sets, you must specify a parameter for each result set on the CALL SQL statement.

To return a single result set and display it using a default report:

Use the following syntax:

```
%function (dtw_sql) function_name () {
    CALL stored_procedure ()
%}
```

For example:

```
 %function (dtw_sql) mystoredproc() {
    CALL myproc
%}
```

To return a single result set and specify a REPORT block for display processing:

Use the following syntax:

```
%function (dtw_sql) function_name () {
     CALL stored_procedure
 %REPORT (resultsetname) {
  ...
 %}
%}
```

For example:

```
%function (dtw_sql) mystoredproc () {
     CALL  myproc
 %REPORT  {
      ...
      %row {  ....   %}
      ...
 %}
%}
```

Alternatively, the following syntax can be used:

```
%function (dtw_sql) function_name () {
    CALL stored_procedure (resultsetname)

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

For example:

```
%function (dtw_sql) mystoredproc () {
    CALL myproc (mytable1)
 %REPORT (mytable1) {
  ...
```

```
  %row {  ....   %}
  ...
 %}
%}
```

To store a single result set in a Net.Data table for further processing:

Use the following syntax:

```
%function (dtw_sql) function_name (OUT tablename) {
   CALL stored_procedure (tablename)
%}
```

For example:

```
%define DTW_DEFAULT_REPORT = "NO"

%function (dtw_sql) mystoredproc (OUT mytable1) {
   CALL myproc (mytable1)
%}
```

Note that DTW_DEFAULT_REPORT is set to `NO` so that a default report is not generated for the result set.

To return multiple result sets and display them using default report formatting:

Use the following syntax:

```
%function (dtw_sql) function_name () {
    CALL stored_procedure (tablename1, tablename2)
%}
```

For example:

```
%function (dtw_sql) mystoredproc () {
    CALL myproc (mytable1, mytable2)
%}
```

To return multiple result sets and have the result sets stored in Net.Data tables for further processing:

Use the following syntax:

```
 %function (dtw_sql) function_name (OUT tablename1, tablename2) {
    CALL stored_procedure (resultsetname1, resultsetname2)
%}
```

For example:

```
%define DTW_DEFAULT_REPORT = "NO"

%function (dtw_sql) mystoredproc (OUT mytable1 mytable2) {
     CALL myproc (mytable1, mytable2)
%}
```

Note that DTW_DEFAULT_REPORT is set to `NO` so that a default report is not generated for the result sets.

To return multiple result sets and specify REPORT blocks for display processing:

Each result set is associated with its own REPORT block. Use the following syntax:

```
%function (dtw_sql) function_name () {
   CALL stored_procedure (resultsetname1, resultsetname2)
 %REPORT (resultsetname1)
   ...
   %row {  ....   %}
   ...
 %}
 %REPORT (resultsetname2)
   ...
   %row {  ....   %}
   ...
 %}
%}
```

For example:

```
%function (dtw_sql) mystoredproc () {
   CALL myproc (mytable1, mytable2)

 %REPORT(mytable1)
  ...
  %row {  ....   %}
  ...
 %}

 %REPORT(mytable2)
  ...
  %row {  ....   %}
  ...
 %}
%}
```

To return multiple result sets and specify different display or processing options for each result set:

You can specify different processing options for each result set using unique parameter names. For example:

```
%function (dtw_sql) mystoredproc (OUT mytable2) {
    CALL myproc (mytable1, mytable2, mytable3)

 %REPORT(mytable1)
 ...
 %row {  ....   %}
 ...
 %}
%}
```

The result set `mytable1` is processed by the corresponding REPORT block and is displayed as specified by the macro writer. The result set `mytable2` is stored in the Net.Data table `mytable2` and can now be used for further processing, such as being passed to another function. The result set `mytable3` is displayed using Net.Data's default report format because no REPORT block was specified for it.

Guidelines and Restrictions for Returning Multiple Result Sets

Use the following guidelines and restrictions when returning multiple result sets from a stored procedure.

Guidelines:

- Specify one REPORT block per result set. The table name parameter specified for the REPORT block must match the corresponding table name parameter on the CALL statement.

- Specify REPORT blocks for multiple result sets in the order in which you want the result sets to be processed.
- To specify default processing when there is not a REPORT block specified for a result set, define DTW_DEFAULT_REPORT = ″YES″. When Net.Data builds the Web page, it displays default reports for result set tables after it displays the reports for result set having REPORT blocks.
- To prevent Net.Data from displaying results sets that do not have REPORT blocks, set DTW_DEFAULT_REPORT = ″NO″.
- When using the DTW_SET_TABLE_IN variable with a stored procedure, the first result set returned from the stored procedure is assigned to the DTW_SET_TABLE_IN table.

Restrictions:

- Multiple REPORT blocks can be used only in stored procedures with the DTW_SQL DB2 language environment.
- Report variables are set for an entire function and affect the processing of all REPORT blocks and the result sets they process. You cannot modify the value of a report variable for individual REPORT blocks.
- The MESSAGE block must be located either before or after a list of REPORT blocks, and not between REPORT blocks.
- Table variables must be defined with the TABLE statement before being used in the stored procedure
- Multiple REPORT blocks cannot be specified for the same result set.
- The maximum number of result sets for a single stored procedure is 32.

Message Blocks

The MESSAGE block lets you determine how to proceed after a function call based on the success or failure of the function call, and lets you display information to the caller of the function. Net.Data uses the following message block process:

1. Net.Data sets RETURN_CODE, a language environment variable, for each function call to a FUNCTION block. RETURN_CODE is not set on a function call to a MACRO_FUNCTION block.
2. When a language environment passes a return code value back to Net.Data, Net.Data sets the value of RETURN_CODE to the return code value.
3. When the function call completes, the MESSAGE block uses the value of RETURN_CODE to determine how to proceed.

A MESSAGE block consists of a series of message statements, each of which specifies a return code value, message text, and an action to take. The syntax of a MESSAGE block is shown in the language constructs chapter of *Net.Data Reference*.

A MESSAGE block can have a global or a local scope. If the MESSAGE block is defined in a FUNCTION block, its scope is local to that FUNCTION block. If it is specified at the outermost macro layer, the MESSAGE block has global scope and is active for all function calls executed in the Net.Data macro. If you define more than one global MESSAGE block, the last one defined is active.

Net.Data uses these rules to process the value of the RETURN_CODE variable from a function call:

1. Check local MESSAGE block for an exact match; exit or continue as specified.

2. If RETURN_CODE is not 0, check local MESSAGE block for `+default` or `-default`; depending on the sign of RETURN_CODE, exit or continue as specified.
3. If RETURN_CODE is not 0, check local MESSAGE block for `default`; exit or continue as specified.
4. Check global MESSAGE block for an exact match; exit or continue as specified.
5. If RETURN_CODE is not 0, check global MESSAGE block for `+default` or `-default`; depending on the sign of RETURN_CODE, exit or continue as specified.
6. If RETURN_CODE is not 0, check global MESSAGE block for `default`; exit or continue as specified.
7. If RETURN_CODE is not 0, issue Net.Data internal default message and exit.

The following example shows part of a Net.Data macro with a global MESSAGE block and a MESSAGE block for a function.

```
%{ 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
  %}
```

If *my_function()* returns with a RETURN_CODE value of 50, Net.Data processes the error in this order:

1. Check for an exact match in the local MESSAGE block.
2. Check for `+default` in the local MESSAGE block.
3. Check for `default` in the local MESSAGE block.
4. Check for an exact match in the global MESSAGE block.
5. Check for `+default` in the global MESSAGE block.

When Net.Data finds a match, it sends the message text to the Web browser and checks the requested action.

When you specify `continue`, Net.Data continues to process the Net.Data macro after printing the message text. For example, if a macro calls `my_functions()` five times and error 100 is found during processing with the MESSAGE block in the example, output from a program can look like this:

```
.
.
.
11 May 1997                 $245.45
13 May 1997                 $623.23
```

```
19 May 1997                  $ 83.02
return code 100 message
22 May 1997                  $ 42.67

Total:                       $994.37
```

Generating HTML in a Macro

Net.Data lets you easily present standard HTML pages on the application user's browser. The following sections describe the HTML and REPORT blocks of the macro and show you how to format HTML in Net.Data macros. See the language constructs chapter in *Net.Data Reference* for syntax information for these blocks.

HTML Blocks

The Net.Data macro file contains HTML blocks and the functions in the HTML blocks that generate HTML output to a Web browser. In a macro file, you must specify at least one HTML block. The contents of the HTML block control the rest of the Net.Data invocation.

Any valid HTML statement can appear in an HTML block. In addition, you can use INCLUDE statements, function calls, and variable references in an HTML block. The following example shows a common use of HTML blocks in a Net.Data macro:

```
%DEFINE DATABASE="MNS96"

%HTML(INPUT){
<H1>Hardware Query Form</H1>
<HR>
<FORM METHOD="POST" ACTION="/cgi-bin/db2www/equiplst.d2w/report">
<dl>
<dt>What hardware do you want to list?
<dd><input type="radio" name="hdware" value="MON" checked>Monitors
<dd><input type="radio" name="hdware" value="PNT">Pointing devices
<dd><input type="radio" name="hdware" value="PRT">Printers
<dd><input type="radio" name="hdware" value="SCN">Scanners
</dl>
<HR>
<input type="submit" value="Submit">
</FORM>
%}

%FUNCTION(DTW_SQL) myQuery() {
SELECT MODNO, COST, DESCRIP FROM EQPTABLE WHERE TYPE=$(hdware)
%REPORT{
<B>Here is the list you requested:</B><BR>
%ROW{
<HR>
$(N1): $(V1)    $(N2): $(V2)
<P>
$(V3)
%}
%}
%}

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

You can invoke the Net.Data macro from an HTML link like the one in the following example:

```
<a href="http://www.ibm.com/cgi-bin/db2www/equiplst.d2w/input">List of hardware</a>
```

When the application user clicks on this link, the Web browser invokes Net.Data, and Net.Data parses the macro file. When Net.Data begins processing the HTML block specified on the invocation, in this case the HTML(INPUT) block, it begins to process the text inside the block. Anything that Net.Data does not recognize as a Net.Data macro language construct, it assumes to be an HTML statement and sends to the browser for display.

After the user makes a selection and presses the Submit button, Net.Data runs the ACTION part of the HTML FORM element, which specifies a call to the Net.Data macro's HTML(REPORT) block. Net.Data then processes the HTML(REPORT) block just as the HTML(INPUT) block was.

Net.Data then processes the `myQuery()` function call, which in turn invokes the SQL FUNCTION block. After replacing the `$(hdware)` variable reference in the SQL statement with the value returned in the input form, Net.Data runs the query. At this point, Net.Data again starts sending the HTML report to the browser, displaying the results of the query according to the HTML statements specified in the REPORT block.

After Net.Data completes the REPORT block processing, it returns to the HTML(REPORT) block, and finishes processing.

Net. Data processes only one HTML block each time it is invoked. However, by using HTML links and forms, you can easily enable the person to start another invocation of Net.Data on another HTML block, all controlled by you.

Report Blocks

Use the REPORT block language construct to format and display data output from a FUNCTION block. This output is typically table data, although any valid combination of HTML tags, macro variable references, and function calls can be specified. A table name can optionally be specified on the REPORT block. If you do not specify a table name, Net.Data uses the table data from the first output table in the parameter list of the FUNCTION block. If you do not specify a table on the FUNCTION block, Net.Data uses the default table data.

The REPORT block has three parts, each of which is optional:

- Header information that contains HTML data displayed once before table row data.
- ROW block that contains HTML and table variables displayed once for each row of the result table.
- Footer information that contains data displayed once after table row data.

To avoid displaying any table output from the ROW block, leave the ROW block empty or omit it entirely.

You can use several Net.Data-provided variables inside the REPORT block to access the data in the Net.Data macro results table. These variables are described in "Table Processing Variables" on page 70. For additional detail, see the Report Variables section in the *Net.Data Reference*.

To provide header and footer information, provide the text before or after the ROW block. Net.Data processes everything it finds before a ROW block and treats it as header information, and treats everything it finds after the ROW block as footer information. As with the HTML block, Net.Data treats everything in the header,

ROW, and footer blocks that is not recognized as macro language constructs as HTML and sends the HTML data to the browser.

You can also use functions and variables in the REPORT block.

To have Net.Data print a default report using preformatted text, do not include the REPORT block in the macro file. The following example shows the default report format:

```
 SHIPDATE   | RECDATE    | SHIPNO   |
------------------------------------
 25/05/1997 | 30/05/1997 | 1495194B |
------------------------------------
 25/05/1997 | 28/05/1997 | 2942821G |
------------------------------------
```

To use the HTML tags instead of the preformatted text, set DTW_HTML_TABLE to YES.

To disable the printing of the a default report, set DTW_DEFAULT_REPORT to NO or by specifying an empty REPORT block. For example:

```
%REPORT{%}
```

The following example shows how you can customize report formats using special variables and HTML tags. It displays the names, phone numbers, and fax numbers from the table `CustomerTbl`:

```
%FUNCTION(DTW_SQL) custlist() {
     SELECT Name, Phone, Fax FROM CustomerTbl
   %REPORT{
<I>Phone Query Results:</I>
<BR>
=====================
<BR>
   %ROW{
   Name: <B>$(V1)</B>
<BR>
Phone: $(V2)
<BR>
Fax: $(V3)
<BR>
------------------------------
<BR>
   %}
   Total records retrieved: $(NUM_ROWS)
   %}
   %}
```

The resulting report looks like this in the Web browser:

Phone Query Results:
==========

Name: **Doen, David**
Phone: 422-245-1293
Fax: 422-245-7383

Name: **Ramirez, Paolo**
Phone: 955-768-3489
Fax: 955-768-3974

Name: **Wu, Jianli**
Phone: 525-472-1234
Fax: 525-472-1234

Total records retrieved: 3

Net.Data generated the report by:

1. Printing *Phone Query Results:* once at the beginning of the report.
2. Giving the variables `V1`, `V2`, and `V3` the values for Name, Phone, and Fax respectively for each row as it is retrieved.
3. Drawing a line after each retrieved row to help readability.
4. Printing the string *Total records retrieved:* and the value for `NUM_ROWS` once at the end of the report.

You can use multiple report blocks for stored procedures to return multiple result sets. Multiple report blocks are supported only for stored procedures when using the DTW_SQL language environment. See "Calling Stored Procedures" on page 77 to learn how to return multiple result sets from a stored procedure.

Conditional Logic and Looping in a Macro File

Net.Data lets you incorporate conditional logic and looping in your Net.Data macros using the IF and WHILE blocks.

Conditional Logic

Use the IF block to do conditional processing in a Net.Data macro. The IF block is similar to IF statements in most high-level languages because it 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 specify IF blocks almost anywhere in a macro and can nest them. The syntax of an IF block is shown in the language constructs chapter in *Net.Data Reference*.

The rules for IF block syntax are determined by the block's position in the macro file. The elements allowed in the executable block of statements of an IF block depend on the location of the IF block itself. Any element that is valid in the block containing the IF block is valid in within that IF block. For example, if you specify an IF block inside an HTML block, any element that is allowed in the HTML block is allowed in the IF block, such as INCLUDE statements and WHILE blocks.

```
%HTML block
...
  %IF block
...
    %INCLUDE
...
    %WHILE
```

Similarly, if you specify the IF block outside of any other block in the declaration part of the Net.Data macro, only those elements allowed outside of any other block (such as a DEFINE block or FUNCTION block) are allowed in the IF block.

```
%IF
...
  %DEFINE
...
  %FUNCTION
```

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

Exception: Do not specify a ROW block in an IF block when the IF block is inside a REPORT block.

Net.Data processes the IF block condition list in one of two ways based on the contents of the terms making up the conditions. The default action is to treat all terms as strings, and to perform string comparisons as specified in the conditions. However, if the following two conditions are met, Net.Data performs a numeric comparison:

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

  Examples of valid strings:

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

  Examples of invalid strings:

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

Net.Data evaluates the IF block at the time it executes the block, which can be different than the time it is originally read by Net.Data. For example, if you specify an IF block a REPORT block, Net.Data does not evaluate the condition list associated with the IF block when it reads the FUNCTION block definition containing the REPORT block, but rather when it calls the function and executes it. This is true for both the condition list part of the IF block and the block of statements to be executed.

Restriction: Net.Data does not support numerical comparison of non-integer numbers.

Example: This example shows a macro file containing IF blocks in other blocks:

```
%{ This macro is called from another macro, passing the operating system
   and version variables in the form data.
%}

%IF (platform == "AS400")
  %IF (version == "V3R2")
     %INCLUDE "as400v3r2_def.hti"
  %ELIF (version == "V3R7")
     %INCLUDE "as400v3r7_def.hti"
  %ELIF (version == "V4R1")
     %INCLUDE "as400v4r1_def.hti"
  %ENDIF
%ELSE
     %INCLUDE "default_def.hti"
%ENDIF

%MACRO_FUNCTION numericCompare(IN term1, term2, OUT result) {
%IF (term1 < term2)
  @dtw_assign(result, "-1")
%ELIF (term1 > term2)
  @dtw_assign(result, "1")
%ELSE
  @dtw_assign(result, "0")
%ENDIF
%}

%HTML(report){
  %WHILE (a < "10") {
    outer while loop #$(a)<BR>
    %IF (@dtw_rdivrem(a,"2") == "0")
      this is an even number loop<BR>
    %ENDIF
    @DTW_ADD(a, "1", a)
  %}
%}
```

Looping Constructs

Use the WHILE block to perform looping in a Net.Data macro. Like the IF block, the WHILE 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. Unlike the IF block, the block of statements can be executed any number of times based on the outcome of the condition test.

You can specify WHILE blocks inside HTML blocks, REPORT blocks, ROW blocks, MACRO_FUNCTION blocks, and HTML IF blocks, and you can nest them. The syntax of a WHILE block is shown in the language constructs chapter of *Net.Data Reference*.

Net.Data processes the WHILE block exactly the same way it processes the IF block, but re-evaluates the condition list each time through the loop. And, like any conditional looping construct, it is possible for processing to go into an infinite loop if the condition is coded incorrectly.

Example: This examples shows a macro file with a WHILE block:

```
%DEFINE loopCounter = "1"

%HTML(build_table) {
%WHILE (loopCounter <= "100") {
  %{ generate table tag and column headings %}
  %IF (loopCounter == "1")
    <TABLE BORDER>
```

```
    <TR>
    <TH>Item #
    <TH>Description
      %ENDIF

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

  %{ generate end table tag %}
  %IF (loopCounter == "100")
       %ENDIF

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

Using Large Objects

Large multimedia and document objects are binary files such as BMP, TIF, GIF, and PostScript files, used for display or printing. You can store these files in DB2 databases on most operating systems and then access them through the SQL language environment for your Web application.

Net.Data supports the of the following types of the following LOBs:

- Binary large objects (BLOBs)
- Character large objects (CLOBs)
- Double-byte large objects (DBLOBs)

LOBs generally contain a file signature in the first few bytes to indicate what type of information the file contains. If Net.Data recognizes a LOB, Net.Data adds the extension to the temporary file and to the Net.Data macro variable representing its name. If you do not have a REPORT block in the macro file, Net.Data adds the .txt extension to CLOB files. Net.Data recognizes the following LOB formats:

- Bitmap (.bmp)
- Graphical image format (.gif)
- Tag image file format (.tif)
- Postscript (.ps) These must be stored as BLOBs, not CLOBs.

Restriction: Net.Data does not recognize other file types and does not support them. Additionally Net.Data does not support UPDATE and INSERT SQL statements for any large objects.

Planning tip: When a query returns a LOB, Net.Data saves it in the directory specified by the HTML_PATH configuration variable. Consider system limitations when using LOBs because they can quickly consume resources. Some LOBs, like audio files, require special hardware and software.

Example 1: Displays a picture inline

```
<IMG SRC="/tmplobs/filename">
<A HREF="/tmplobs/filename">filename</A>
```

Example 2: In the following example, the application user must click on the file name to invoke the viewer because the application uses a .WAV files. Net.Data does not recognize this file type so an EXEC variable is used to append the extension to the file.

```
%DEFINE{
docroot="/usr/lpp/internet/server_root/html"
rename=%EXEC "rename $(docroot)$(V3) $(docroot)$(V3).wav"
%}

%FUNCTION(DTW_SQL) queryData() {
SELECT Name, IDPhoto, Voice FROM RepProfile
%REPORT{
<P>Here are the images you selected:<P>
%ROW{
$(rename)
$(V1) Voice sample <IMG SRC="$(V2)">
<A HREF="$(V3.wav")>Voice sample</A><P>
%}
%}
%}

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

The `queryData` function returns the following HTML:

```
<P>Here are the images you selected:<P>
Kinson Yamamoto
<IMG SRC="/tmplobs/p2345n1.gif">
<A HREF="/tmplobs/p2345n2.wav">Voice sample</A><P>
Merilee Lau
<IMG SRC="/tmplobs/p2345n3.gif">
<A HREF="/tmplobs/p2345n4.wav">Voice sample</A><P>
```

The REPORT block in the previous example uses the implicit table variables V1, V2, and V3.

- V1 is a person's name, which is plain text.
- V2 is a photo of the person in a .GIF file. The image is shown inline. Net.Data includes the temporary LOB directory and .GIF extension automatically.
- V3 is a sample of the person's voice in a .WAV file. When Net.Data encounters an unrecognized file format, such as a .WAV file, it writes the file into the temporary LOB directory without a file extension. This example shows how to handle this file type by adding the extension using an EXEC variable. When the variable *$(V3)* is resolved, it has the path `/tmplobs/` added before the file name. For example, `/tmplobs/sound2a`. One way to deal with such files is to write a REXX program to change the slashes to back slashes and rename the files. The voice sample is played when the application user clicks on Voice sample.

Not all Web browsers support graphics and sound. Special hardware and software, such as a sound card and driver, might be required to support the features described here.

Chapter 6. Using Built-in Functions

Net.Data provides a large set of built-in functions to simplify Web page development. These functions are already defined by Net.Data, so you do not need to define them in a FUNCTION block. You can simply call these functions anywhere in the macro where a user-defined function can be called.

Built-in functions can return their results in three ways. You can tell how each one returns its results by its prefix:

- **DTW_, DTWF_, and DTWR_:** The results of the call are returned in an output parameter or no result is returned. (**DTWF**_ is the prefix for flat file functions. **DTWR**_ is the prefix for Web registry functions.)
- **DTW_r, DTWF_r, and DTWR_r:** The results of the function call replace the function call in the macro, in the same way the value of the RETURNS keyword replaces the function call for a user-defined function which has specified a RETURNS keyword.
- **DTW_m:** Multiple results are returned in the parameters passed to the function.

Some built-in functions do not have each type. For more information, see *Net.Data Reference*.

The following list provides a high-level overview of the Net.Data built-in functions. Use these functions to perform general purpose, math, string, word, or table manipulation functions. See *Net.Data Reference* for descriptions of each function with syntax and examples.

General Purpose Functions This set of functions help you develop Web pages by altering data or accessing system services. You can use them to query and set environment variables, use HTML escape codes, and get other useful information from the system.

Math Functions These functions perform mathematical operations, letting you calculate or alter numeric data. Besides standard mathematical operations, you can also perform modulus division, specify a result precision, and use scientific notation.

String Functions These functions let you manipulate characters within strings. You can change a string's case, insert or delete characters, assign a string value to another variable, plus other useful functions.

Word Functions These functions let you manipulate words in character strings. Most of these functions work the same as string functions, but on entire words. For example, they let you count the number of words in a string, remove words, search a string for a word.

Table Functions You can use these functions to generate reports or forms using the data in a Net.Data table variable. Table variables contain an array of values and their associated column names. They provide a convenient way to pass groups of values to a function.

Flat File Functions Use the flat file interface (FFI) functions to open,

	read, and manipulate data from flat file sources (text files), as well as store data in flat files.
Web Registry Functions	Use the Web registry functions to maintain registries and the entries they contain. A Web registry is a file with a key maintained by Net.Data to allow you to add, retrieve, and delete entries easily.

Chapter 7. Using Language Environments

Net.Data supplies language environments that you use to access data sources and to execute application programs containing business logic. For example, the SQL language environment lets you pass SQL statements to a DB2 database, and the REXX language environment lets you invoke REXX programs. You can also use the SYSTEM language environment to execute a program or issue a command.

With Net.Data, you can add user-written language environments in a pluggable fashion. Each user-written language environment must support a standard set of interfaces that are defined by Net.Data and must be implemented as a dynamic link library (DLL) or shared library. You must add an ENVIRONMENT statement to the Net.Data initialization file to associate a DLL with the language environment written by you. Net.Data loads and executes a DLL the first time it encounters a function call for a FUNCTION block that specifies the language environment name. Subsequent function calls for FUNCTION blocks that specify the same language environment name merely cause Net.Data to execute the DLL, because DLLs are only loaded once.

Figure 21 shows the relationship between the Web server, Net.Data, and the Net.Data language environments.

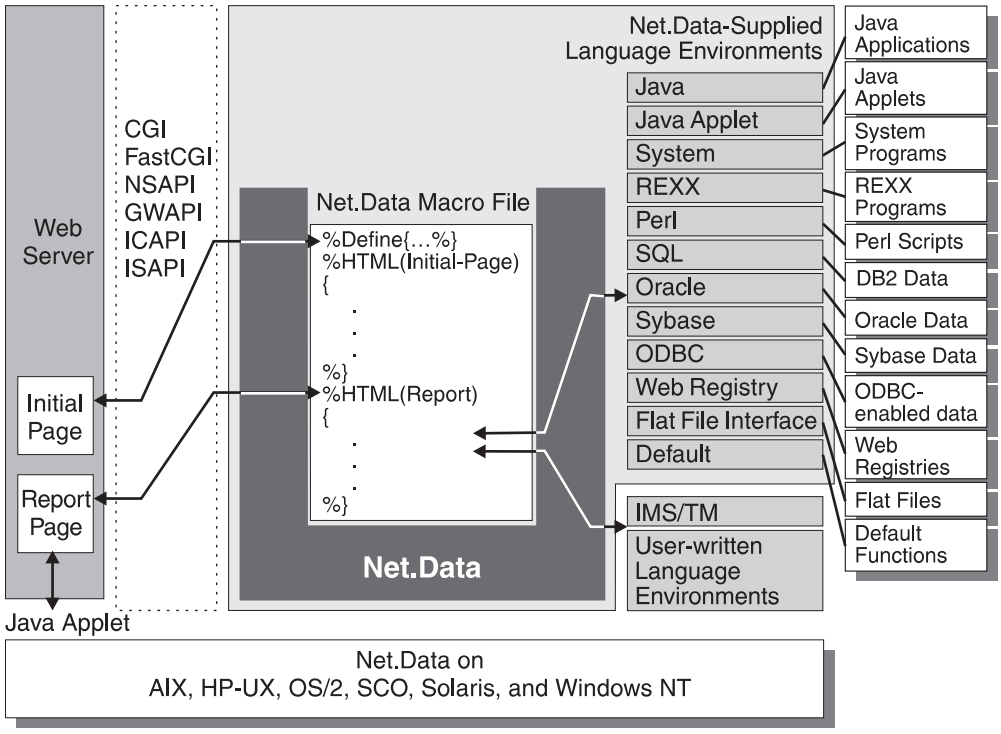


Figure 21. The Net.Data Language Environments

For complete details on Net.Data-supplied language environments and on how to create a user-written language environment, see the *Net.Data Language Environment Reference*.

Chapter 8. Invoking Net.Data with Java Servlets and JavaBeans

Net.Data provides Java-based classes that you can use to invoke Net.Data macro files, run Net.Data functions, or execute SQL statements through Net.Data within any Java-enabled operating system supported by Net.Data.

This chapter describes the concepts and tasks for:

Net.Data Servlets

Net.Data provides servlets and NetObjects Fusion plug-ins with Net.Data functions that can be used in a Java environment. With servlets and plug-ins you can:

- Run Net.Data macro files from both a URL and as a Server-Side-Include (SSI)
- Run Net.Data functions from both a URL and as an SSI
- Use NetObjects Fusion (NOF) to manage your macro files, providing better integration with your Web site management and an easy-to-use graphical user interface to develop simple macros. See "Appendix C. Using NetObjects Fusion NOF Plug-ins with Net.Data Servlets" on page 145 to learn how to use NOF plug-ins.

This section describes the following servlet topics:

About Net.Data Servlets

Net.Data provides servlets to help you develop and manage macros using the Java environment. Servlets are Java classes that perform a role similar to that of CGI programs or Web server API plug-ins. Servlets are used by a Java servlet-enabled Web server to build HTML pages. Servlets do not have their own graphical user interface, but their classes can be dynamically loaded locally, or from across the network, and can be called using a URL address (remotely) or by a class name (locally). Servlets are available for Windows NT and AIX operating systems.

The Net.Data servlets are a Java-based wrapper that run a Net.Data Version 2 macro file or direct request using a native DLL file. These servlets allow you to run an existing macro file (MacroServlet) or a single function (FunctionServlet). Net.Data Version 2 is required to use these servlets. Net.Data servlets can run the database or non-database language environments and can run with Live Connection.

The servlets come with an API for use with your applications. The servlet APIs are documented with Net.Data. See the <*inst_dir*>/servlets/NetDataServlets.jar file for API documentation.

Net.Data provides two servlets:

Macro Servlet (com.ibm.netdata.servlets.MacroServlet)
Executes an existing Net.Data macro file.

Using the macro servlet is similar to running a Net.Data macro file through the CGI-BIN interface, except you run the macro file through a Java servlet. The macro servlet requires that Net.Data Version 2 or higher be installed.

The advantages of using the macro servlet include:

- SQL queries are run through ODBC using Live Connection Manager for increased performance.
- You can run macro files through Server-Side-Includes (SSI) to imbed multiple macros in your HTML file.

The macro servlet also allows native access to heterogeneous databases, such as DB2, Oracle, and Sybase, as well as various language environments, such as Perl, REXX, and Java.

Function Servlet (com.ibm.netdata.servlets.FunctionServlet)
Executes the Net.Data function or SQL statement through a servlet interface, such as %FUNCTION DTW_SQL(). See "Invoking Net.Data without a Macro File (Direct Request)" on page 51 for more information.

The function servlet requires that Net.Data Version 2 be installed.

Setting Up Servlets

See your Web server documentation for instructions on registering and using servlets. The Net.Data servlets are contained in the Net.Data <*inst_dir*>/servlets/NetDataServlets.jar file. Your Web server might require that you add <*inst_dir*>/servlets/NetDataServlets.jar to your CLASSPATH environment variable.

Running Net.Data Servlets

The Net.Data servlets can be run either from a URL or as an SSI within an HTML file. You can use the NetObjects Fusion plug-ins to incorporate the Net.Data servlets into your NOF site. The following sections discuss how to modify and run the servlets by typing in the syntax for the servlet. See "Appendix C. Using NetObjects Fusion NOF Plug-ins with Net.Data Servlets" on page 145 to learn how to modify and run the servlets with NetObjects Fusion.

Running the Macro Servlet

From within an HTML file, enter the servlet parameters using one of the following syntax options:

1. URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet
  ?MACRO=macro_value&BLOCK=block_value&parmnn=valuenn
```

   For example:

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=my_macro
  &BLOCK=my_block&field1=custno
```

2. SSI:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
     <param name="MACRO" value="macro_value">
     <param name="BLOCK" value="block_value">
     <param name="parmnn" value="valuenn">
 </servlet>
```

For example:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
     <param name="MACRO" value="my_macro.d2w">
     <param name="BLOCK" value="report">
     <param name="field1" value="custno">
 </servlet>
```

Parameters:

macro_value
The fully qualified path to an existing Net.Data macro file

block_value
The name of the HTML block in the specified Net.Data macro file to execute; the default is `report` (optional).

parmnn
Any additional parameters that your macro file requires, such as

```
<param name="field1" ...
```

valuenn
Any additional values that your macro file requires, such as

```
... value="custnum"
```

HTMLPATH parameter: If you get an error message referring to a missing HTMLPATH parameter, add the HTMLPATH parameter to your servlet invocation command:

- URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet
?MACRO=macro_name&BLOCK=block_value&htmlpath=html_path&parmnn=valuenn
```

For example:

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=my_macro
  &BLOCK=my_blockhtmlpath=e:\html&field1=custno
```

- SSI:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
     <param name="MACRO" value="macro_value">
      <param name="BLOCK" value="block_value">
      <param name="htmlpath" value="html_path">
     <param name="parmnn" value="valuenn">
 </servlet>
```

For example:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
<param name="MACRO" value="my_macro">
<param name="BLOCK" value="my_block">
<param name="htmlpath" value="e:\html">
<param name="field1" value="custno">
</servlet>
```

INBUFLEN and OUTBUFLEN parameters: If your input to the macro file is larger than 1 KB, you must specify the INBUFLEN parameter. If your macro file results are

larger than 32 KB, you must specify the OUTBUFLEN parameter. Failure to specify these parameters when required can result in unpredictable results.

- URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet
?MACRO=macro_name&BLOCK=block_value&INBUFLEN=input_buffer_size
&OUTBUFLEN=output_buffer_size&parmnn=valuenn
```

  For example:

```
http://myserver/servlet/com.ibm.netdata.servlets.MacroServlet?MACRO=my_macro
  &BLOCK=my_blockINBUFLEN=3K&OUTBUFLEN=48K&field1=custno
```

- SSI:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
     <param name="MACRO" value="macro_value">
      <param name="BLOCK" value="block_value">
<param name="INBUFLEN" value="input_buffer_size">
<param name="OUTBUFLEN" value="output_buffer_size">
<param name="parmnn" value="valuenn">
 </servlet>
```

  For example:

```
<servlet code="com.ibm.netdata.servlets.MacroServlet">
<param name="MACRO" value="my_macro">
<param name="BLOCK" value="my_block">
<param name="INBUFLEN" value="3K">
<param name="OUTBUFLEN" value="48K">
<param name="field1" value="custno">
</servlet>
```

Running the Function Servlet

The function servlet can invoke Net.Data using direct request to execute either a function (such as a REXX function) or an SQL statement. The parameters you specify for the servlet depend on whether you are executing a function or an SQL statement. From within an HTML file, enter the servlet parameters using one of the following syntax options:

1. For a URL:
   - ***To invoke a function:***

     ```
     http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
       ?LANGENV=lang_env_name&FUNC=program_name&parmnn=valuenn
     ```

     For example:

     ```
     http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
       ?LANGENV=DTW_REXX&FUNC=my_rexx&field1=custno
     ```

   - ***To invoke an SQL statement:***

     ```
     http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
        ?LANGENV=database_lang_env_name&SQL=SQL_statement
        &DATABASE=database_name&parmnn=valuenn
     ```

     For example:

     ```
     http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
     ?LANGENV=DTW_SQL&SQL=select+*+from+myTable&DATABASE=CELDIAL
     ```

2. SSI:
   - ***To invoke a function:***

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
     <param name="LANGENV" value="lang_env_name">
      <param name="FUNC" value="program_name">
      <param name="parmnn" value="valuenn">
 </servlet>
```

For example:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_REXX">
<param name="FUNC" value="myREXX">
<param name="field1" value="custno">
</servlet>
```

- ***To invoke an SQL statement:***

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
     <param name="LANGENV" value="lang_env_name">
      <param name="SQL" value="SQL_stmt_name">
      <param name="DATABASE" value="database_name">
     <param name="parmnn" value="valuenn">
 </servlet>
```

For example:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_SQL">
<param name="SQL" value="select * from employee">
<param name="DATABASE" value="CELDIAL">
</servlet>
```

Parameters:

lang_env_name
: The Net.Data language environment (such as DTW_SQL, DTW_REXX) being called to process the function, SQL statement, or stored procedure. This parameter requires that FUNC or SQL be used.

program_name
: The name of the program which contains the function to be executed. For example, `my_rexx`, where `my_rexx` is the name of a REXX executable. Specify input parameters for the function with the *parmnn* and *valuenn* parameters.

database_name
: The name of the database associated with the DATABASE parameter. The specified

SQL_stmt_name
: An SQL statement or stored procedure name that accesses a database. For example, `"select * from employee"`. Specify input parameters for the SQL statement or stored procedure with the *parmnn* and *valuenn* parameters.

parmnn
: Any additional parameters that your macro file requires, such as `<param name="field1" ...`.

valuenn
: Any additional values that your macro file requires, such as `... value="custnum"`.

HTMLPATH parameter: If you get an error message referring to a missing HTMLPATH parameter, add the HTMLPATH parameter to your servlet invocation command:

- URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=lang_env_name&FUNC=program_name&htmlpath=html_path
&parmnn=valuenn
```

For example:

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=DTW_REXX&FUNC=my_rexx&htmlpath=e:\html&field1=custno
```

- SSI:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
     <param name="LANGENV" value="lang_env_name">
      <param name="SQL" value="SQL_stmt_name">
      <param name="htmlpath" value="html_path">
     <param name="parmnn" value="valuenn">
 </servlet>
```

For example:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_SQL">
<param name="SQL" value="select * from employee">
<param name="htmlpath" value="e:\html">
<param name="field1" value="custno">
<param name="DATABASE" value="SAMPLE">
</servlet>
```

Where *html_path* specifies the path to the Web server root HTML directory; for example: `htmlpath=e:\html`.

INBUFLEN and OUTBUFLEN parameters: If your input to the macro file is larger than 1 KB, you must specify the INBUFLEN parameter. If your macro file results are larger than 32 KB, you must specify the OUTBUFLEN parameter. Failure to specify these parameters when required can result in unpredictable results.

- URL:

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet
?LANGENV=lang_env_name&FUNC=program_name&INBUFLEN=input_buffer_size
&OUTBUFLEN=output_buffer_size&parmnn=valuenn
```

For example:

```
http://myserver/servlet/com.ibm.netdata.servlets.FunctionServlet?LANGENV=DTW_REXX
  &FUNC=my_rexxINBUFLEN=3K&OUTBUFLEN=48K&field1=custno
```

- SSI:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
     <param name="LANGENV" value="lang_env_name">
      <param name="FUNC" value="program_name">
<param name="INBUFLEN" value="input_buffer_size">
<param name="OUTBUFLEN" value="output_buffer_size">
<param name="parmnn" value="valuenn">
 </servlet>
```

For example:

```
<servlet code="com.ibm.netdata.servlets.FunctionServlet">
<param name="LANGENV" value="DTW_REXX">
<param name="FUNC" value="my_rexx">
<param name="INBUFLEN" value="3K">
<param name="OUTBUFLEN" value="48K">
<param name="field1" value="custno">
</servlet>
```

Net.Data JavaBeans

Net.Data provides JavaBeans that can be used in a Java environment without having a Web server running. A JavaBean is an object-oriented programming interface that lets you build reuseable applications or program building blocks. These objects can be used in a network on Java-enabled operating system.

Using a native Net.Data DLL, the JavaBean invokes Net.Data, populating the return code and a string containing the Net.Data output (results). Because JavaBeans use a native DLL, you do not have to have a Web server running to use Net.Data functions.

Design tip: The results returned by the Net.Data JavaBeans are whatever your macro file or function returns; in general, this is HTML. Consider passing the results to an HTML-like JavaBean which understands HTML and displays the results.

With JavaBeans you can:

- Run Net.Data macro files
- Run SQL statements through Net.Data

This section describes the following JavaBean topics:

About Net.Data JavaBeans

Net.Data provides JavaBeans to help you develop and manage macros using the Java environment. JavaBeans are Java objects that provide the following interface:

- When used in a JavaBean development environment (such as Lotus BeanMachine), you use the provided customizer to hook together the desired components to process and display the results of a macro file or SQL statement and produce a Java applet.
- When using the API, you can use the JavaBeans to provide Net.Data functionality to your own Java applet or application. The API documentation is in *<inst_dir>*/beans/NetDataBeans.jar.

Net.Data provides two types of JavaBeans:

Net.Data Macro JavaBean
: Provides a Java-based interface for executing an existing Net.Data macro through Net.Data.

Net.Data SQL JavaBean
: Provides a Java based interface for executing an SQL statement through Net.Data.

The Net.Data JavaBeans are Java-based wrappers that run through Net.Data using a native DLL file. Both require Net.Data Version 2 or higher and JDK Version 1.1 or higher to be installed.

Setting Up and Running the Net.Data JavaBeans

This section describes how to set up and run Net.Data JavaBeans using a JavaBean development tool, such as Bean Machine. Steps for using development tools are generic so that you can use the tool of your choice.

- "Setting Up the Macro Bean"
- "Running the Macro Bean"
- "Setting Up the SQL Bean"
- "Running the SQL Bean" on page 105

Setting Up the Macro Bean

The Net.Data Macro bean, com.ibm.netdata.beans.NetDataMacro, lets you use Java to run an existing macro file. To use this bean, you need to specify Net.Data properties for the bean so that it can work with the macro file.

To set up the Net.Data macro JavaBean with a JavaBean development tool:

1. Add or import the <*inst_dir*>/beans/NetDataBeans.jar file into your JavaBean development tool.
2. Using the development tool's customizer interface, set the following input properties:

   Macro Specifies the name of the existing macro file to execute. For example: MyMacro.mac

   Block Specifies the name of the HTML block section to execute; the default is report.

   HTML path
   Specifies the path to the Net.Data db2www.ini file.

   Parameters
   Specifies the parameter name and values to use when running macro.

   Syntax:

   ```
   name1=value1&nameN=valueN
   ```

Running the Macro Bean

To run the Net.Data macro JavaBean with a JavaBean development tool:

- Select the run or execute action provided by your JavaBean development tool to run the macro.
- After the macro has run, you can reference the following output properties:

  RC Specifies the return code returned from Net.Data.

  Results
  Specifies the data returned from the execution of the Net.Data macro file.

Setting Up the SQL Bean

The Net.Data SQL bean, com.ibm.netdata.beans.NetDataSQL, lets you use Java to run an SQL statement through Net.Data. To use this bean, you need to specify Net.Data properties for the bean so that it can work with the macro file.

To set up the Net.Data macro JavaBean with a JavaBean development tool:

1. Add or import the NetDataBeans.jar file into your JavaBean development tool.
2. Using the development tool's customizer interface, set the following input properties:

   Language environment
   Specifies the language environment to use; the default is `DTW_SQL`.

   SQL Specifies the SQL statement to run; the default is `select * from employee`.

   DATABASE
   Specifies the database to use; the default is `SAMPLE`.

   HTML path
   Specifies the path to the Net.Data `db2www.ini` file.

   Parameters
   Specifies the parameter name and values to use when running the SQL statement.

   Syntax:

   ```
   name1=value1&nameN=valueN
   ```

Running the SQL Bean

To run the Net.Data macro JavaBean with a JavaBean development tool:

- Select the run or execute action provided by your JavaBean development tool to run the macro.
- After the SQL statement has run, you can reference the following output properties:

  RC Specifies the return code returned from Net.Data.

  Results
  Specifies the data returned from the SQL statement.

Chapter 9. Net.Data Caching

Caching helps you to improve response times for the application user. Net.Data stores results from a request to the Web server locally for quick retrieval, until it is time to refresh the information. This chapter describes Net.Data caching concepts, tasks, and restrictions.

About Web Caching

Many software components perform caching for Web applications. Here are some examples of caching applications:

- A Web browser saves Web pages and related objects such as image and audio files and Java applets, locally, in memory or on disk to save network time when its user repeatedly accesses the same pages.
- A Web proxy server cache saves Web pages and related objects on a local server, near a group of users, to reduce network access time to remote Web servers, for example to reduce the number of times the Web servers retrieve requested items. A Web proxy server cache also enables efficient sharing of commonly accessed pages between multiple users.
- A Web server caches frequently retrieved pages and related objects in memory to save disk access time when users repeatedly retrieve the same pages.
- A database management system caches data items, which are usually held on disk, in memory to save disk access time when repeatedly retrieving the same data items.

All these components perform their caching independently but the overall result is improved response times for users. In order to determine when to refresh a cached item, the Web components (browser, proxy server, and Web server) usually take into consideration various options including:

- The browser and server configuration options
- The contents of the HTTP headers returned with the Web pages and related items from the Web server, in particular the expiry date information

About Net.Data Caching

Net.Data itself provides its own caching function for frequently accessed pages and related data items generated by Net.Data macros. By delivering a page from the Net.Data cache, you save the time required to run a Net.Data macro and to access a database in order to create the page.

You can use one Cache Manager per server. **Recommendation:** Use one Cache Manager for many instances of Net.Data, and multiple caches per Cache Manager.

Figure 22 shows that Net.Data uses a Cache Manager to manage the caching of HTML output from a macro file. This output can include data from a database.

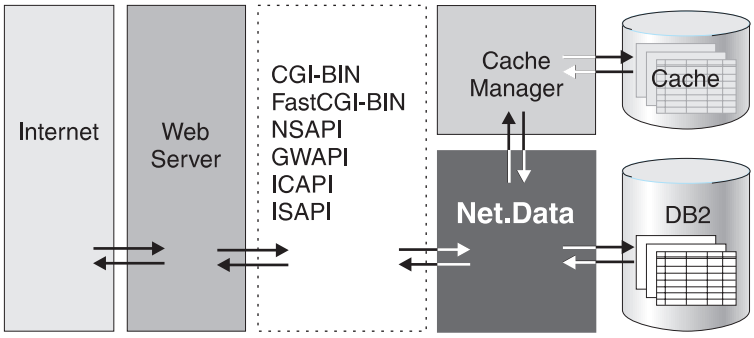


Figure 22. Net.Data Caching

Net.Data Caching Terminology

Net.Data documentation uses the following terms to describe Net.Data caching.

cache A type of memory that contains recently accessed data, designed to speed up subsequent access to the same data. The cache is often used to hold a local copy of frequently-used data that is accessible over a network. In Net.Data, local memory that contains Net.Data-generated HTML Web pages for reuse by the Net.Data macro. By having the pages stored in the cache, Net.Data does not have to regenerate the information in the cache. Each cache is managed by the Cache Manager, which can be responsible for multiple caches and can server multiple instances of Net.Data.

cache ID
A string that identifies a particular cache.

Cache Manager
The program that manages caching for one machine. It can manage multiple caches.

Cache Manager configuration file
The file containing the settings used by Net.Data to determine settings for logging, tracing, cache size, and other options. It contains settings for a Cache Manager and all the cache files managed by a particular Cache Manager. The file name is `cachemgr.cnf` when packaged with Net.Data.

Net.Data Caching Concepts

Depending on how many HTTP servers you have on your system, and whether each HTTP server runs its own copy of Net.Data (using separate Net.Data configuration files), you can have all the copies of Net.Data be associated with one Cache Manager or multiple Cache Managers. One Cache Manager can support a number of caches in memory, each cache has a cache identifier called a *cache ID*. Figure 23 shows one Cache Manager working with multiple macro files and managing two caches.

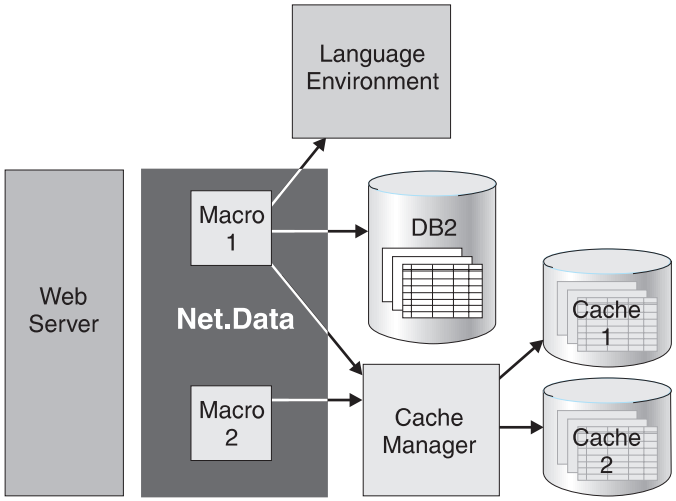


Figure 23. Cache Manager Works with Multiple Macro Files and Caches

Any number of items, known as *cached pages*, can be placed in a cache. Each cached page has a unique identifier, for example a Uniform Resource Locator (URL). A page is a segment of or a complete HTML page.

When Net.Data receives a request for cached data (for example, from the built-in function DTW_CACHE_PAGE), the following steps are taken:

1. Net.Data connects to the Cache Manager.
2. Net.Data checks to see if the data is cached.
   - If the data is cached and not expired, Net.Data requests the page from the Cache Manager, sends it to the browser, and stops executing the macro.
   - If the data is not cached, Net.Data continues processing the macro, then send the generated HTML page to the Web browser and to the Cache Manager, where it is cached.
3. Net.Data disconnects from the Cache Manager

The Cache Manager caches HTML output when the macro file successfully completes processing, ensuring that only successfully generated Web pages are cached. The data is not cached until after it has been sent to the browser, and the data that the user sees is the same data that is cached.

When Net.Data encounters an error or exits prematurely from the macro, the Cache Manager:

- Does not accept partial or defective pages
- Preserves existing pages in the cache

Net.Data Caching Restrictions

Net.Data caching has the following restrictions:

Security
Security is not provided by the Cache Manager. For example, if one database user runs a macro and caches a pages of database results. Another database user can retrieve the cached page.

Direct request
Direct request invocation of Net.Data cannot use Net.Data caching.

Net.Data Caching Interfaces

Net.Data provides a flexible set of interfaces for you to use when configuring and setting up caching for you application. Table 6 describes the various options for using the Net.Data caching features and where these features are described.

Table 6. Net.Data Cache Interfaces

Interface	Description	Go to ...
Cache Manager configuration options	You can specify a number of options for the Cache Manager such as logging and tracing, in the Cache Manager stanza of the Cache Manager configuration file.	“Defining the Cache Manager” on page 112
Cache configuration options	Within a single instance of Net.Data’s Cache Manager, you can define a number of caches to hold the cached items. Each cache has its own set of characteristics, such as size and location, and the cache ID. These characteristics are defined in the cache stanza in the Cache Manager configuration file. Each stanza is identified by the cache ID.	“Defining a Cache” on page 114
Net.Data initialization options	If Net.Data and the corresponding Cache Manager run on separate systems, then you specify the Cache Manager system and port number in the Net.Data initialization file.	“Cache Manager Configuration Variables” on page 10
Net.Data cache built-in functions	You can manipulate the contents of a Net.Data cache using the Net.Data cache built-in functions. Specify the cache ID in the appropriate macro function to select the cache with the most appropriate characteristics.	See the built-in functions chapter of *Net.Data Reference*

Planning for the Cache Manager

When planning your use of the Net.Data cache functions you must consider:

- What pages would benefit from caching and the performance improvements that would be gained
- When to cache the items
- When to refresh the items in the cache and the refresh methods to use

To use Net.Data caching, you need to complete the following steps, which require knowing how you want to use caching.

Recommendation: Before embarking on a major application that uses caching, we strongly urge you to plan and prototype your application prior to putting it into production.

- Install Net.Data, which includes the caching function.
- Configure the Cache Manager. See “Configuring the Cache Manager and Net.Data Caches” on page 112.
- Determine how you are going to put the Net.Data application into production.
- Check the various Net.Data cache logs to determine whether improvements in the use of the cache and the way it is configured should be made.

Cache Errors

The Cache Manager does not cache Web pages when Net.Data encounters internal errors that cause it to exit the macro file before processing is complete. Cache Manager does not cache pages that are incomplete or contain Net.Data errors. These types of errors include macro syntax errors and SQL errors.

Pages with errors can be cached when:

- Net.Data encounters an error and Net.Data continues to execute the macro file because of a CONTINUE directive in a message block and terminates normally.
- Errors occur outside of the Net.Data’s error determination scope, such as database rollbacks.

Cache Identifiers

You need to plan for two types of identifiers when designing caching for your application.

- **Identifier for a cache:** This identifier is the cache ID and specifies the name in the configuration file stanza that defines the cache. You can use many approaches for classifying and naming your caches. For example, you can name the cache by application. You could have a cache for each of your Net.Data applications, giving each cache a name that is derived from the Net.Data macro it serves.
- **Identifier for the cached page:** This identifier is the cached page ID and specifies the name of the page to be cached. The cached page ID can be any string, such as an URL address. You specify the identifier with the DTW_CACHE_PAGE() built-in function. See the built-in functions chapter of *Net.Data Reference* for syntax and examples.

Configuring the Cache Manager and Net.Data Caches

The Cache Manager manages one or more caches in your system. Each of these caches contains the contents of dynamically generated HTML pages. To configure the Cache Manager and each of the caches, you update the keyword values in the Cache Manager configuration file, `cachemgr.cnf`.

The Cache Manager configuration file contains two types of stanzas: the Cache Manager stanza and the Cache Definition stanza. The following steps describe how to customize these two types of stanzas for your application.

Defining the Cache Manager

Define the Cache Manager stanza by specifying values for the allowed keywords. All of the keywords are optional; you do not need to specify them unless you don't want to accept the default value.

To define the Cache Manager:

1. Specify the name of the Cache Manager log file. The log shows activity for all transactions for all caches and is provided for debugging and problem analysis.

   The default is to write messages to the console.

   Syntax:

   `log=`*`path`*

   Where *path* is the path and file name of the cache file.

   Tip: To specify a log file for each cache, use the **tran-log** keyword in the Cache Definition stanza.
2. Specify the TCP/IP port number used by the Cache Manager for incoming requests. This port number is used only for contacting the Cache Manager from a remote machine.

   This value must match the port number specified by the CACHE_PORT configuration variable in the Net.Data initialization file. The default value is determined in the following manner:

   a. The Cache Manager checks the path */etc/services* for the value associated with the name `ibm-cachemgrd`. If this value is found, the Cache Manager uses the value. If it is not found, it uses the next method.

   b. The Cache Manager uses the default port, 7175.

   Syntax:

   `port=`*`port_number`*

   Where *port_number* is a unique TCP/IP port number.
3. Specify the maximum length of time in seconds that the Cache Manager should allow a pending read to be left active. If this time is exceeded, the Cache Manager drops the connection.

   The default is 30 seconds.

   Syntax:

   `connection-timeout=`*`seconds`*

   Where *seconds* is the number of seconds used for the length of time a pending read should be active.

4. Specify whether to log messages.

   The default is **no** or **off**.

   Syntax:

   ```
   logging=yes|on|no|off
   ```

   Where:

   yes|on
   Indicates that logging is required

   no|off Indicates that logging should not be performed.

5. Specify whether to wrap the log.

   The default is **no**. If specified as **yes**, the current log is closed when it reaches its maximum size (see **log-size**, below), the file has a file type of .old, and a new log is opened. Only one generation of the log file is maintained (existing .old files are overwritten).

   Syntax:

   ```
   wrap-log=yes|no
   ```

   Where:

   yes Specifies to wrap the log.

   no Specifies not to wrap the log.

6. Specify the maximum size, in bytes, to which a log is allowed to grow, if wrap-log is specified.

   The default is 64000.

   Syntax:

   ```
   log-size=bytes
   ```

   Where *bytes* is the number of bytes of the maximum size.

7. Specify the level of messages to be written to the log. These values are set on when included in the *trace_flag_definitions* list and do not have any settings.

   The default is to log only the Cache Manager start up and shut down messages.

   Syntax:

   ```
   trace-flags=trace_flag_definitions
   ```

   Where:

   D_ALL
   Enables all trace flags.

   D_NONE
   Disables all trace flags

Example: Trace flag that specifies all trace flags are enabled:

```
trace=flags=D_ALL
```

Stanza example: A valid Configuration Manager stanza:

```
cache-manager {
log=/u/cached/logs/cached.log
port=7177
connection-timeout=0
logging=yes
```

```
wrap-log=yes
log-size=64000
trace-flags=D_ALL
}
```

Defining a Cache

Define the Cache Definition stanza by specifying values for the allowed keywords. Most keywords are optional and do not have to be specified unless you don't want to use the default value.

To define a cache:

1. Specify the path and directory name that are to hold the cache pages. On start up, the file system containing this directory must be at least as large as the value of **fssize** (see below); otherwise the cache is not started. This value can be specified as an absolute path name or as a relative path name that corresponds to the path in which the Cache Manager was started.

   Required.

   Syntax:

   `root=`*`path_name`*

   Where:

   path_name
   Is the absolute or relative name of the path and directory where the cache pages are stored.

2. Specify whether the current cache is active when the Cache Manager starts.

   Not required; the default is **yes**. If set to **no**, the cache is defined in the Cache Manager but not activated. You can activate it later with the **cacheadm** command.

   Syntax:

   `caching=yes|no`

   Where:

   yes Indicates that the cache is to be active when the Cache Manager starts.

   no Indicates that the cache is not to be active when the Cache Manager starts.

3. Specify the maximum space to be used in the file system by pages in the current cache. When the maximum amount of space is exceeded, the Cache Manager deletes enough pages, starting with the oldest, to bring the total space occupied by the cache within the limit. You can effectively disable automatic purging of entries by setting this value to a large number; however, if the physical file system space is exceeded, attempts to add new pages to cache fail.

   Not required; the default is 0 (no caching to disk).

   Syntax:

   `fssize=`*`nn`*`B|`*`nn`*`KB|`*`nn`*`M`

   Where:

   *nn***B** Is the number of bytes; for example 5000B.

   *nn***KB** Is the number of kilobytes; for example 640KB.

*nn***MB** Is the number of megabytes; for example 30MB.

4. Specify the maximum amount of memory to be used by all of the pages in this cache. When the maximum amount of memory is exceeded, the Cache Manager deletes enough pages, starting with the oldest, to bring the total memory occupied by the cache within bounds. You can effectively disable automatic purging of pages by setting this to a large number; however, if the **cachemgrd** process consumes too much memory, the operating system may terminate it.

   Not required; the default is 1MB.

   Syntax:

   `mem-size=`*nn*`B|`*nn*`KB|`*nn*`MB`

   Where:

   *nn***B** Is the number of bytes; for example 5000B.

   *nn***KB** Is the number of kilobytes; for example 640KB.

   *nn***MB** Is the number of megabytes; for example 30MB.

5. Specify the maximum length of time an page can be held in the cache. When this value is exceeded, the Cache Manager marks the page as expired but does not delete the page unless the **fssize** (if it is cached on disk) or **memsize** (if it is cached in memory) limits are reached. The Cache Manager deletes pages that are marked as expired before all other pages if **memsize** or **fssize** limits are reached. You can disable **lifetime** checking with the **check_expiration** keyword.

   Required: No; the default is 5 minutes.

   Syntax:

   `lifetime=`*`time_length`*

   Where:

   *nn***S** Is the number of seconds; for example, 600S.

   *nn***M** Is the number of minutes; for example, 20M.

   *nn***H** Is the number of hours; for example, 30H.

6. Specify whether to mark cache pages as expired and to perform lifetime checking.

   Not required; the default is **yes**, with a default lifetime length of 60 seconds. This value can also be set to a length of time, indicating a **yes** value and declaring a maximum length of time an item may be held in cache. When set to **no**, cache pages are never marked expired and lifetime checking is not performed.

   Syntax:

   `check-expiration=yes|`*nn*`S|`*nn*`M|`*nn*`H|no`

   Where:

   yes Indicates that the Cache Manager performs lifetime checking and cache pages are marked as expired.

   *nn***S** Is the number of seconds; for example, 600S.

   *nn***M** Is the number of minutes; for example, 20M.

   *nn***H** Is the number of hours; for example, 30H.

no Indicates that the Cache Manager does not perform lifetime checking and cache pages are not marked as expired.

7. Specify the maximum amount of space a cached page can occupy within the memory cache. If an page is too large for memory, the file cache is checked. If adequate space exists, the Cache Manager stores the cache page in the file cache, instead. If the page does not fit in the file cache, the caching attempt fails. If the page is smaller than datum_memory_limit value (cacheobj-memory-limit), but if the cache doesn't have enough space, the oldest cache pages are deleted from the memory cache to accommodate the new page.

   Not required; the default is 1KB.

   Syntax:

   `datum-memory-limit (cacheobj-memory-limit)=`*nn*`B|`*nn*`KB|`*nn*`MB`

   Where:

   *nn***B** Is the number of bytes; for example 5000B.

   *nn***KB** Is the number of kilobytes; for example 640KB.

   *nn***MB** Is the number of megabytes; for example 30MB.

8. Specify the maximum amount of space a cache page can occupy within the file cache. If an page is smaller than datum_disk_limit, but no space remains in the file cache, the oldest cache pages are deleted from the file cache to accommodate the new page.

   Not required; the default is 1KB.

   Syntax:

   `datum-disk-limit (cacheobj-space-limit)=`*nn*`B|`*nn*`KB|`*nn*`MB`

   Where:

   *nn***B** Is the number of bytes; for example 5000B.

   *nn***KB** Is the number of kilobytes; for example 640KB.

   *nn***MB** Is the number of megabytes; for example 30MB.

9. Specify the time between creation of statistics records. If set to 0, no statistics records are written.

   Not required; the default is 0 (no statistics).

   Syntax:

   `stat-interval = `*nn*`S|`*nn*`M|`*nn*`H`

   Where:

   *nn***S** Is the number of seconds; for example, 600S.

   *nn***M** Is the number of minutes; for example, 1M.

   *nn***H** Is the number of hours; for example, 3H.

10. Specify the name of the path and file that are to be used for logging statistics for the current cache.

    Required when the value for **stat-interval** is greater than 0.

    Syntax:

    `stat-files=`*filename*

    Where *filename* is the path and name of the logging statistics file.

11. Specify whether statistics counters should be reset to 0 each time they are written to the log file.

    Not required; the default is **yes**.

    Syntax:

    ```
    reset-stat-counters=yes|no
    ```

    Where:

    yes Resets the statistics counters.

    no Does not reset the statistics counters.

12. Define the path and file name to hold the transaction log for each cache. Cache transaction log files are separate from Cache Manager log files, which are used to log overall Cache Manager activity.

    Required; if not specified, a transaction log for the cache is not created.

    Syntax:

    ```
    tran-log=filename
    ```

    Where *filename* is the path and file name of the transaction logs for each cache.

13. Specify whether to turn on transaction logging for the cache when the Cache Manager first starts up. This parameter is ignored unless a valid transaction log file is specified via the tran-log parameter. You can activate transaction logging while the Cache Manager daemon is running using the **cacheadm** command if a valid tran-log value has been specified in the Cache Manager configuration file.

    Not required; the default is **no**.

    Syntax:

    ```
    tran-logging=yes|on|no|off
    ```

    Where:

    yes|on
    Indicates that logging is required.

    no|off Indicates that logging should not be performed.

14. Specify whether the transaction log should be wrapped.

    Not required; the default is **yes**. If specified as **yes**, the current log is closed when it reaches the maximum size (see **tran-log-size**), has file type of .old, and a new log is opened. Only one generation of the log is maintained (existing .old files are overwritten).

    Syntax:

    ```
    wrap-tran-log=yes|no
    ```

    Where:

    yes Indicates to wrap the log.

    no Indicates to not wrap the log.

15. Specify the maximum size in bytes to which a transaction log is allowed to grow, if **wrap-tran-log** is specified.

    Not required; the default is 64000.

    Syntax:

    ```
    tran-log-size=bytes
    ```

Where *bytes* is the number of bytes of the maximum size.

Stanza example: A valid Cache Definition stanza for a cache:

```
test1
{
caching=on
fssize=5MB
mem-size=10MB
lifetime=6000000
check-expiration=150
datum-memory-limit=5KB
datum-disk-limit=500KB
stat-interval=60
reset-stat-counters=no
root=/u/cached/chaches/cache0
stat-files=/u/cached/logs/chache0.stats
tran-log=/u/cached/logs/cache0.log
tran-logging=yes
wrap-tran-log=yes
tran-log-size=100k
}
```

Starting and Stopping the Cache Manager

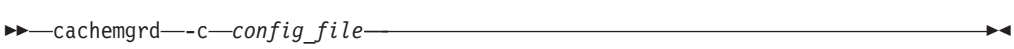
The following sections describe how to start and stop the Cache Manager.

- "Starting the Cache Manager"
- "Stopping the Cache Manager"

Starting the Cache Manager

Use the `cachemgrd` command to start the Cache Manager daemon.

Syntax:



Parameters:

cachemgrd
The command keyword.

config_file
Specifies the name of the file where the Cache Manager and each of the caches managed by the Cache Manager are defined. The configuration file shipped with the Net.Data product is `cachemgr.cnf`.

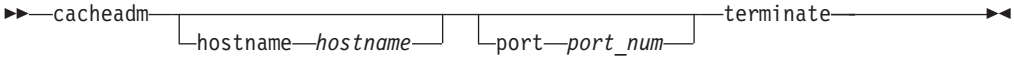
Example:

```
cachemgrd -c myconfig.cfg
```

Stopping the Cache Manager

Use the `cacheadm` command to stop the Cache Manager.

Syntax:



Parameters:

cacheadm
The command keyword.

hostname
Specifies the name of the machine where the cache is running, if it is different from the machine where the `cacheadm` command is issued.

port_num
Specifies the cache port number, if the number is different from the default (7175).

terminate
Specifies to stop the Cache Manager.

Example:

```
cacheadm hostname host1 port 7178 terminate
```

Caching Web Pages

You can use the DTW_CACHE_PAGE built-in function to cache a Web page. When Net.Data sees the DTW_CACHE_PAGE function in the macro file, it contacts the Cache Manager and begins saving the HTML output for the macro file in memory. After Net.Data successfully processes a macro, the HTML output is sent to the browser and the Cache Manager caches the output in one transaction as shown in Figure 24.

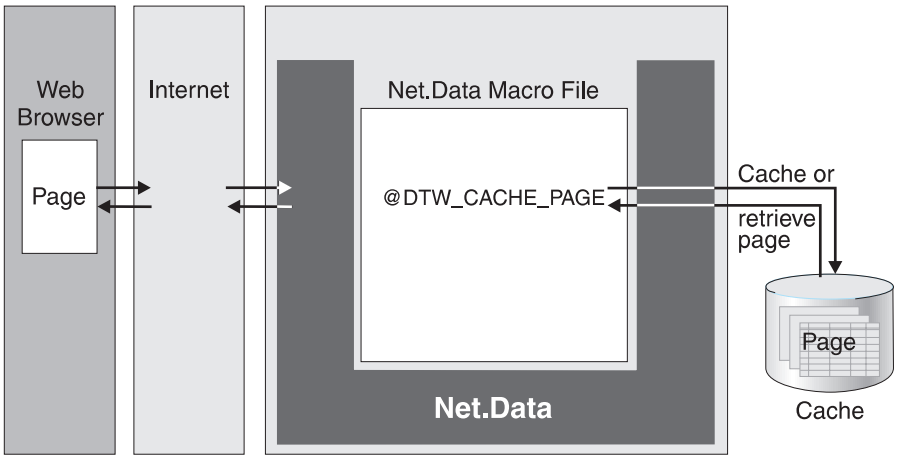


Figure 24. DTW_CACHE_PAGE Function Initiates Caching

Caching a Page

Specify Net.Data-generated pages to be written to the cache using the Net.Data DTW_CACHE_PAGE() built-in function.

The DTW_CACHE_PAGE() function caches all of the output from the macro file following the function statement, once it determines that the page does not already exist in the cache or has expired. If the page does not exist in the cache or is older than the specified age, Net.Data sends the output page back to the browser, generates new output pages from the macro execution, and stores the page in the cache.

If the Cache Manager finds the cached page and it is still current, it displays the cache contents and Net.Data exits out of the macro. This behavior ensures that no unnecessary processing is done after the Web page has been retrieved from the cache.

Performance tip: Place DTW_CACHE_PAGE() as the first, or one of the first statements in a macro file, to minimize the cost of executing the macro file.

To cache a page:

1. In the HTML block of a macro file, prior to the HTML coding, insert the following function statement:

   ```
   @DTW_CACHE_PAGE("cache_id", cached_page_id, "age", status)
   ```

   Use this function to specify that Net.Data is to cache all HTML output from the macro that follows this statement. Place this statement early in the macro file if you want to cache all HTML output.

   Parameters:

   cache_id
   : A string identifying the cache in which the page will be placed. You can associate cache IDs with macros or groups of macros.

   cached_page_id
   : A string containing an identifier used to identify the page in the cache in a subsequent @DTW_CACHE_PAGE cache request, for example the page's URL.

   age
   : A string variable containing a length of time in seconds that specifies when a page is considered out-of-date. If the requested page has been in the cache for longer than the value of *age*, Net.Data executes the macro, regenerates the page, and caches the generated page, replacing the out-of-date page. If the requested page has been in the cache for less than or the same as the value of age, Net.Data retrieves the page from the cache and sends it to the browser. In this case, Net.Data ends macro execution immediately.

   status
   : A string variable returned by Net.Data to indicate whether or not the page was cached successfully.

Example:

```
%HTML(cache_example) {
 %IF (customer == "Joe Smith")
 @DTW_CACHE_PAGE("mymacro.d2w", "http://www.mypage.org", "-1", status)
 %ENDIF
 ...
 <html>

<head>
 <:title>This is the page title</title>
 </head>
```

```
<body>
 <center>
 <h3>This is the Main Heading</h3>
 <p>It is $(time). Have a nice day!
 </body>

</html>
 %}
```

Advanced Caching: Determining Dynamically Whether to Cache

The DTW_CACHE_PAGE() function initiates caching from its location in the macro file. Normally, you place the function at the beginning of the macro file for better performance and to ensure that all HTML output is cached.

For advanced caching applications, you can place the DTW_CACHE_PAGE() function in the HTML output sections when you need to make the decision to cache at a specific point during processing, rather than at the beginning of the macro file. For example, you might need to make the caching decision based on how many rows are returned from a query or function call.

Example: Places the function in the HTML block because the decision to cache depends on the expected size of the HTML output

```
%DEFINE { ...%}

...

%FUNCTION(DTW_SQL) count_rows(){
  select count(*) from customer
%REPORT{
 %ROW{
  @DTW_ASSIGN(ALL_ROWS, V1)
%}
%}
%}

%FUNCTION(DTW_SQL) all_customers(){
  select * from customer
%}

%HTML(OUTPUT) {
<html>
<head>
 <title>This is the customer list
 </head>
 <body>

@count_rows()

 %IF ($(ALL_ROWS) > "100")
 @DTW_CACHE_PAGE("mymacro.d2w", "http://www.mypage.org", "-1", status)
%ENDIF

@all_customers()

 </body>
 </html>
%}
```

In this example, the page is cached or retrieved based on the expected size of the HTML output. HTML output pages are considered cache-worthy only when the database table contains more than 100 rows. Net.Data always sends the text in the

OUTPUT block, `This is the customer list`, to the browser after executing the macro; the text is never cached. The lines following the function call, `@count_rows()`, are cached or retrieved when the conditions of the IF block are satisfied. Together, both parts form a complete Net.Data output page.

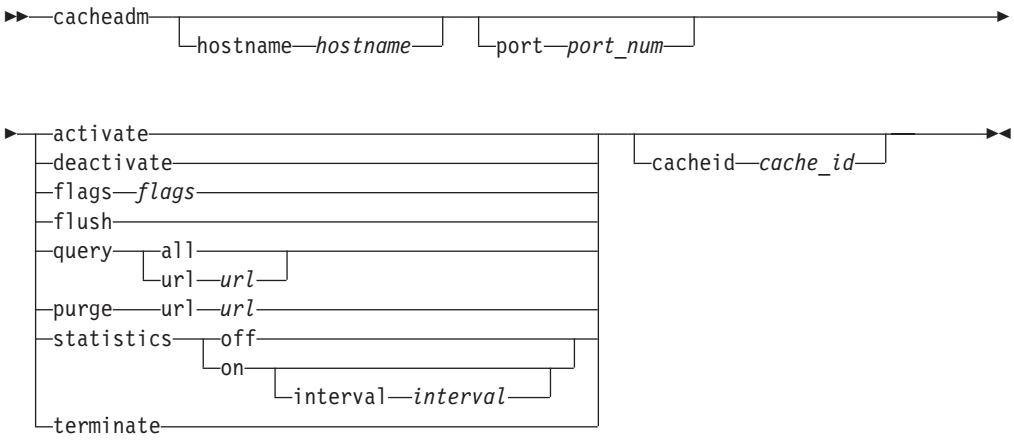
The CACHEADM Command

Use the CACHEADM command for the following tasks:

- Stopping the Cache Manager
- Flushing a specific cache
- Querying a specific cache
- Enabling or disabling logging
- Logging flags
- Starting and stopping statistics gathering

All parameters can be abbreviated to the minimum unique set of characters.

Syntax:



Parameters:

activate
Activates a specified cache. If the cache is already active, Cache Manager does nothing.

cache_id
A string variable identifying the cache in which the page is located. For example: `cache1`.

deactivate
Deactivates a specified cache. If the cache is already inactive, Cache Manager does nothing. All pending operations are completed and no new ones accepted. When the last operation is completed, the Cache Manager marks the cache as inactive.

flags
Specifies whether the listed flags should be toggled on or off.

D_ALL
Turn on all trace flags.

D_NONE
Turn off all trace flags.

flush
Flushes a cache, specified by the *cache_id* parameter, which is required by this parameter. This parameter unconditionally deletes all items from the specified cache.

hostname
Specifies the name of the machine where the cache is running, if it is different from the machine where the `cacheadm` command is issued. For example: `myhost`.

port_num
Specifies the cache port number, if the number is different from the default (7175). This number must be unique within the system.

purge
Specifies that a specific page to be purged from cache. If *url* is specified, Cache Manager purges the page with a key matching *url*. If dependency is specified, Cache Manager purges all items with the associated dependency and writes their keys to `stdout`, the standard output stream.

query
Returns caching data, depending on the parameters you specify:

- Returns information about a cache, if only the cache ID is specified.
- Returns information about a specific cached page if *url* is specified.
- Returns information about all pages if `all` is specified.

  Other programs use `all` option to format or interpret the results. Each line contains the following information:

  – Page key
  – Page age
  – Page length
  – Page creation date
  – Page expiration date
  – Date the page was last referenced

  All dates are in standard UNIX integer time format.

  Performance tip: The option `cache query all` can impact performance and should be used sparingly.

statistics
Enables or disables logging of statistics gathering for a specific cache and requires *cache_id* parameter. If an interval is specified with the `statistics` parameter set to `on`, Net.Data sets or resets the interval between updates to the specified number of seconds.

terminate
Specifies to stop the Cache Manager.

tranlogging
Enables or disables transaction logging for a specific cache and requires the

cache_id parameter. This parameter takes affect only if a valid transaction log for the cache is specified in the Cache Manager configuration file with the `tran-log` parameter.

url The Universal Relative Location (URL) address that specifies the location of the file on a Web server. For example: `http:www.ibm.com/mydir/page1`.

The Cache Log

Several types of statistics regarding internal operation are kept and optionally written to the cache log. A separate log for each cache can be maintained, or all statistics may be written to the same log. This section discusses the following cache log topics:

- "Configuring the Log"
- "Cache Log Format"

Configuring the Log

To log statistics, you must configure the Cache Manager configuration file.

To configure the log:

Specify the `stat-files` and `stat-interval` keywords in the cache stanza of the Cache Manager configuration file.

You can modify statistics settings without stopping, reconfiguring, and restarting the Cache Manager.

To modify statistics gathering settings:

Specify the `cacheadm statistics` command. Note however, that changes made with the `cacheadm statistics` command are not saved when the Cache Manager is restarted.

Cache Log Format

The statistics log is a plain ASCII file that can be processed or imported by spreadsheets or database programs. Three types of records are written:

- Initialization records document the startup of statistics gathering for a particular cache. These records have the following format:

  mm/dd/yy hh:mm:ss id `Initialization: interval` *n* `seconds`

  Where:

mm/dd/yy	Is the month, day, and year when statistics gathering starts
hh:mm:ss	Is the hour, minute, and second when statistics gathering starts
id	Is the name of the cache associated with the record
n	Is the collection interval

- Termination records document the termination of statistics gathering for a particular cache. These records have the following format:

  mm/dd/yy hh:mm:ss id `Termination`

Where:

mm/dd/yy Is the month, day, and year when statistics gathering stops

hh:mm:ss Is the hour, minute, and second when statistics gathering stops

id Is the name of the cache associated with the record

- Statistics records are a blank-delimited set of numbers showing activity within the cache. These records have the following format:

```
mm/dd/yy hh:mm:ss id statistics
```

Where:

mm/dd/yy Is the month, day, and year when statistics gathering is created

hh:mm:ss Is the hour, minute, and second when statistics gathering is created

id Is the name of the cache associated with the record

<*statistics***>** Is a blank-delimited list of statistics gathered for this cache as specified in Table 7:

Table 7. List of Statistics

Field Number	Contents	Description	Counters Reinitialized to Zero
1	reads	Number of read operations performed against the cache	Yes
2	writes	Number of write operations performed against the cache	Yes
3	closes	Number of close operations performed on objects in the cache	Yes
4	open read	Number of open-read operations performed on objects in the cache	Yes
5	open write	Number of open-write operations performed on objects in the cache	Yes
6	open write query	Number of open-write-query operations performed on objects in the cache	Yes
7	read hits	Number of read hits on objects in the cache	Yes
8	write hits	Number of write hits on objects in the cache	Yes
9	write query hits	Number of write query hits on objects in the cache	Yes
10	initializations	Number of new sessions established with this cache	Yes
11	terminations	Number of sessions terminated with this cache	Yes
12	purges	Number of objects deleted from this cache	No
13	memory used	Amount of memory used by objects in the memory portion of the cache	No
14	disk used	Amount of disk space used by objects in the disk portion of the cache	No
15	memory available	Amount of memory still available for use by objects in the memory portion of the cache	No
16	disk available	Amount of disk space still available for use by objects in the disk portion of the cache	No

Table 7. List of Statistics (continued)

Field Number	Contents	Description	Counters Reinitialized to Zero
17	memory object count	Number of objects in the memory portion of the cache	No
18	file object count	Number of objects in the disk portion of the cache	No
19	session count	Number of sessions currently active against the cache	No

Chapter 10. Improving Performance

Improving performance is an important part of tuning your system. This chapter discusses strategies for improving performance. The following topics are discussed:

Improving Performance using the Web Server APIs

You can improve performance by invoking Net.Data with Web server APIs instead of CGI. When Net.Data runs in the Web server API mode, Net.Data executes as a thread inside the Web server's process, eliminating the overhead of invoking Net.Data as a CGI process. With Web server APIs, Net.Data runs as multiple threads within the server's process.

By default, the Web server invokes Net.Data as a CGI program, with each Net.Data process running in a separate process. To improve performance, Net.Data provides configuration options for Web server APIs.

Net.Data supports the Web APIs in the following list, depending on your operating system:

GWAPI plug-in and ICAPI plug-in
: The Lotus Domino Go Webserver API plug-in as the follow-on to the IBM Internet Connection Secure Sever plug-in

ISAPI plug-in
: Microsoft Internet Server API plug-in

NSAPI plug-in
: Netscape Server API plug-in

See the operating system reference appendix in *Net.Data Reference* to determine which Web server APIs are supported for your operating system. See "Configuring Net.Data for Use with the Web Server APIs" on page 26 to learn how to configure Net.Data and the Web server for use with APIs.

Consideration: Using the Web server APIs provides improved performance, without application isolation. Because Net.Data runs in a multi-threaded mode, errors with user-developed language environments, improper invocations, or even database outages can cause problems with the Web server and potentially bring it down. When deciding whether to use one of the Web server APIs versus CGI or FastCGI, determine whether the higher priority for your application is performance or application isolation.

Requirements:

- If running Net.Data in GWAPI, ICAPI, ISAPI or NSAPI mode, you must restart your Web server so that the Web server can reload Net.Data and run it as a process.
- If you make changes to the initialization file after the Web server invokes Net.Data in API mode, you must restart the Web server. Any changes to the Net.Data initialization file (`db2www.ini`) have no effect. In API mode, Net.Data reads the initialization file only once to reduce the performance overhead.
- When running in API mode, the Oracle and Sybase language environments require Live Connection.

To invoke the Web server APIs:

For ICAPI and GWAPI:

Syntax:

`http://`*`server_name`*`/CGI-BIN/db2www/`*`macro_name`*`/`*`html_block`*

Parameters:

server_name
The name of the server.

macro_name
The relative path name of your macro file under the directory specified by MACRO_PATH.

html_block
The name of the HTML block in the macro file to be processed.

Example:

```
http://myserver/CGI-BIN/db2www/mymacro.d2w/report
```

For ISAPI:

Syntax:

`http://`*`server_name/server_HTML_root_directory/dll_name/macro_name/html_block`*

Parameters:

server_name
The name of the server.

server_HTML_root_directory
The Web server HTML root directory name.

dll_name
Net.Data's ISAPI .dll file name, `dtwisapi.dll`.

macro_name
The relative path name of your macro file under the directory specified by MACRO_PATH.

html_block
The name of the HTML block in the macro file to be processed.

Example:

```
http://myserver/scripts/dtwisapi.dll/mymacro.d2w/report
```

For NSAPI:

Syntax:

```
http://server_name/macro_name/html_block
```

Parameters:

server_name
The name of the server.

macro_name
The relative path name of your macro file under the directory specified by MACRO_PATH. The extension of the macro file, for example, `.d2w`, must be defined in the Web server configuration file. See “Configuring Net.Data for Use with the Web Server APIs” on page 26 for more information.

html_block
The name of the HTML block in the macro file to be processed.

Example:

```
http://myserver/mymacro.d2w/report
```

Improving Performance with FastCGI

FastCGI provides improved performance with the reliability of CGI-BIN. Using FastCGI allows you to run macros with the speed of the API servers, with the more reliable method of using separated memory space. Net.Data invocation default is to run with CGI.

You can use Net.Data with FastCGI on all servers that support FastCGI.

See “Configuring Net.Data for FastCGI” on page 23 to learn how to configure for FastCGI.

You can tune FastCGI to run the appropriate amount of processes to handle the number of incoming requests with the process configuration parameter. For example, one customer averaged 100 requests per second and each request took a half second to process. They set the process parameter to 50.

FastCGI is supported by the following language environments:

- SQL
- Oracle 7.2, 7.3, 8.0
- ODBC
- Sybase
- REXX
- Perl

Requirement: When running in FastCGI mode, the Oracle and Sybase language environments require Live Connection.

To tune the number of simultaneous processes:

1. Open the configuration file where the configuration parameter for processes is defined.
   - For Apache, this is the `httpd.conf` file.
   - For ICS or Lotus Domino Go Webserver, this is the `lgw_fcgi.conf` file.

2. Change configuration parameter value that specifies the number of processes:
   - For Apache: `Process=num`.
   - For ISC: `NumProcess=num`.

   Where *num* is the number of processes.

Improving Performance with Connection Management

Net.Data provides a component called Live Connection to manage database and Java virtual machine connections. Live Connection maintains persistent connections to improve performance. Some Net.Data actions require a large start-up time. For example, before a database query can be issued, the process must identify itself to the DBMS and connect to the database. This is often a significant portion of the processing time needed for Net.Data macros that access a database. Another example of an expensive start up is a Java virtual machine that is needed to run Java applications (but not Java applets). Because of the way CGI programs operate, these start-up costs are paid on every request to the Web server. Net.Data provides Live Connection on the OS/2, Windows NT, and UNIX operating systems to maintain persistent connections.

The following sections describe Live Connection.

About Live Connection

Live Connection can dramatically improve performance by eliminating start-up overhead. The savings come from continuously running one or more processes that perform the start up functions. These processes then wait to service requests. You can run Live Connection if you use Net.Data as a CGI or FastCGI program, or as a Web server API plug-in.

Live Connection consists of Connection Manager and cliettes. *Cliettes* are processes that the Connection Manager starts, and stay active while the server is running. Cliettes process data and communicate with Net.Data language environments that you specify in the initialization file with the keyword CLIETTE. Each type of cliette handles a specific language environment function, such as the DB2 cliette, which connects to the DB2 database and sets up operations to perform SQL calls before any Net.Data macros are processed by Net.Data. The executable file is named in the Live Connection configuration file, `dtwcm.cnf`. Figure 25 on page 131 shows the interaction between Live Connection, the macro file, and the language environments.

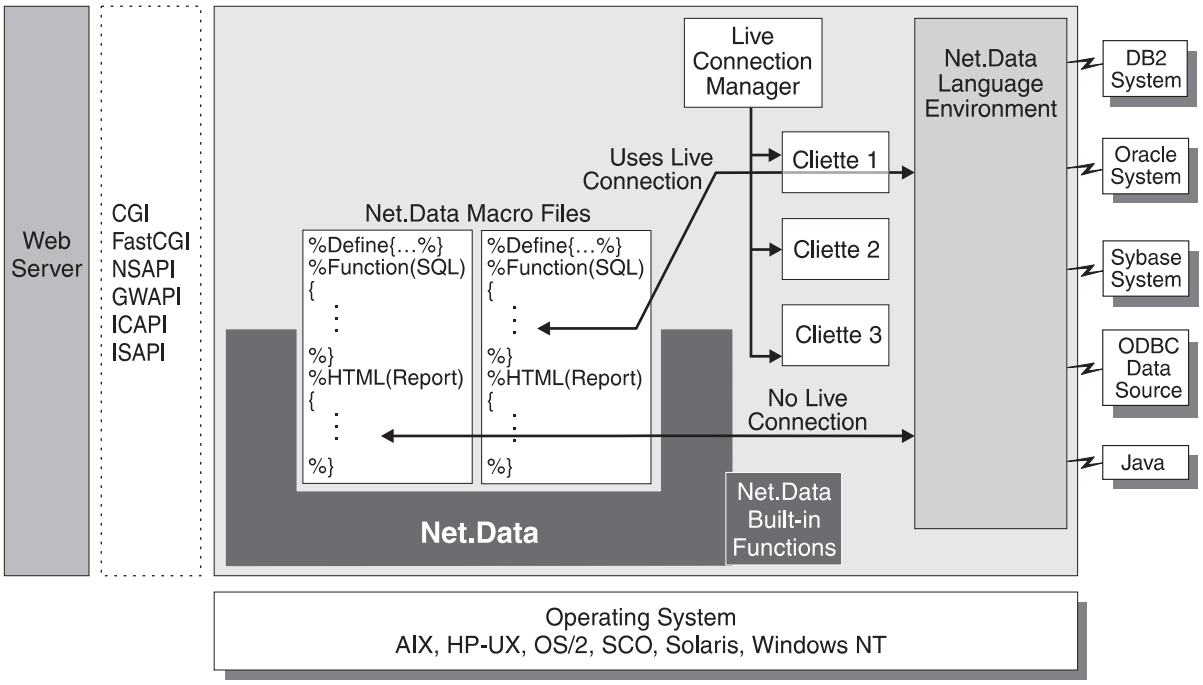


Figure 25. Live Connection with Cliettes

The following sections describe Live Connection in more detail. To learn how to configure Live Connection, see “Configuring Live Connection” on page 18.

Live Connection Advantages

The main advantages to using Live Connection are:

- **Improved performance**

  Reusing connections is more efficient than making new connections. Generally, if you request small SQL statements (for example, simple queries on a database with fewer than 100 000 rows), or if your database connection is difficult (for example, remote servers), the connect time is significant.

- **Multiple database access**

  Live Connection enables one Net.Data macro connection to multiple databases at the same time. This is possible because each database has unique cliettes, and therefore Net.Data simply communicates with multiple cliettes.

Should I Use Live Connection?

You can use Live Connection in CGI, FastCGI, or API mode to communicate with your database. In addition, you might benefit from Live Connection if your application requires data from multiple databases.

Live Connection, used together with an API plug-in, improves performance for many systems, depending on their load and configuration. You should experiment with your own system to determine the configuration that works best for you.

Many applications can improve performance without using Live Connection by using the ACTIVATE DATABASE or START DATABASE command to save time establishing database connections. See your database’s documentation for details

on the command your database uses. Also check your operating system's documentation to see if there are additional steps to help improve performance.

Requirment: The Oracle and Sybase language environments require Live Connection when running in CGI, FastCGI, and API modes.

Starting the Connection Manager

The Connection Manager is a separate executable file that is shipped with Net.Data and named `dtwcm`. Start the Connection Manager when you start the Web server.

When you start Connection Manager, it reads a configuration file and starts a group of processes. In each process, the Connection Manager begins the execution of a particular cliette. To learn how to configure Live Connection, see "Configuring Live Connection" on page 18.

To start the Connection Manager with Windows NT and OS/2:

1. From the command line, change to the <*inst_dir*>\connect\ directory.
2. Enter `dtwcm`.

Where <*inst_dir*> is the Net.Data installation directory.

To start the Connection Manager with AIX:

1. From the command line, change to the `/usr/lpp/internet/db2www/db2/` directory.
2. Enter `dtwcm`.

To start the Connection Manager with the messages option:

By default, Connection Manager messages are suppressed. Use the -d option when starting Connection Manager if you want Connection Manager messages to be displayed.

From the command line, enter: `dtwcm -d`

After you use the -d option, you have to restart Connection Manager in order to suppress the messages again.

To automatically start Connection Manager as a Windows NT service:

On Windows NT, you can specify to have Connection Manager start as an Windows NT service, instead of from the command line. Running Connection Manager as an Windows NT service allows Connection Manager to be automatically started each time the machine is started.

Tip: Start Connection Manager from the command line before setting it up to start automatically to insure that the Live Connection configuration file is correct.

1. From the Windows NT task bar, select **Start->Settings->Control Panel ->Services**.
2. Select **Net.Data Connection Manager** and then click the **Startup** button.
3. Select **Automatic startup type** and then click on **OK**.

Net.Data and Live Connection Process Flow

After you've configured and started the database, Web server, and Connection Manager, Net.Data processing typically involves these steps when Live Connection is enabled:

1. The Web server receives a request and starts either a FastCGI, CGI or API process to run Net.Data.
2. Net.Data starts processing the Net.Data macro.
3. When Net.Data encounters a function call that uses Live Connection, it determines what type of cliette is needed from the initialization file. For DB2, the cliette type is often a name based on the DB2 database name, such as DTW_SQL:CELDIAL.
4. Net.Data asks the Connection Manager for a cliette of that type.
5. The Connection Manager looks for available cliettes of that type. If none is available, the Connection Manager puts the request on a queue and processes it when the right cliette type is available.
6. When a cliette becomes available, the Connection Manager tells Net.Data how to communicate with the cliette.
7. Net.Data asks the cliette to process the function.
8. This process is repeated from step 3 until the Net.Data macro processing is complete.
9. All cliettes are released.

If a cliette is specified in the initialization file but the Connection Manager is not running, Net.Data loads the DLL and processes the macro. If you use an API, you are likely to receive errors, and you should start the Connection Manager.

Improving Performance with the Net.Data Caching

You can improve the performance of your system by caching frequently requested Net.Data Web pages in a cache. By retrieving a page from the cache, you can reduce the processor time used because the Net.Data macro is not interpreted, nor are any Net.Data language environment calls executed. If the Net.Data pages are generated from information held in databases or files then you are also likely to reduce disk IO accesses. As a result of these savings, you might also find that throughput and response time can be reduced.

Net.Data caching provides multi-threaded management of multiple caches within a single process. Caching works well in a networked environment in which a single cache is shared among multiple processes on multiple machines. Net.Data caching uses the Cache Manager component to manage Net.Data caches.

Depending the sizes of the caches you define and the memory use of your existing system, you might require additional memory in order to avoid additional paging.

See "Chapter 9. Net.Data Caching" on page 107 to learn how to set up and use Net.Data caching.

Net.Data Error Logging: Performance Considerations

Net.Data provides an error log so that you can monitor errors or performance problems in your Net.Data system.

When using the Net.Data error log, you might notice an impact on the performance of your system if many messages are being written to the error logs. For example, each time a user accesses a macro that Net.Data cannot find, Net.Data passes a message as output to the error log.

To reduce the impact on performance, check the logging level of the error log set in a Net.Data macro with the DTW_LOG_LEVEL keyword. If the level is set to WARNING, consider reducing the level to ERROR for a small performance gain, or to OFF for a larger performance gain.

Optimizing Math Functions

You can improve the performance of your math functions by using the DTW_OPTIMIZE_MATH configuration variable. This variable provides C language style formatting of mathematical functions, allowing functions to be processed faster. When set to YES, Net.Data uses the C-style formatting. NO is the default. See “DTW_OPTIMIZE_MATH: Optimize Math Functions Variable” on page 12 to learn how to configure this variable.

Syntax:

```
DTW_OPTIMIZE_MATH=NO|YES
```

Consideration: Net.Data Version 1 used REXX style formatting for math functions. Your output might be different when using the DTW_OPTIMIZE_MATH configuration variable. Trailing blanks are not displayed. If consistent output with Net.Data Version 1 macro files is important to your application, consider the merits of consistency with performance.

Chapter 11. Logging Net.Data Error Messages

Net.Data writes error messages to the Net.Data error log file, `netdata.log`. The maximum size of the error log is fixed by Net.Data at 500 KB, approximately 3000 log entries.

You can browse the error log file, or archived copies, periodically to determine whether there are problems in your Net.Data system.

This chapter discusses the following logging topics:

Planning for Monitoring the Log

When logging errors, you need to plan for the following issues:

- Determining disk space:

  If you plan to use error logging, you must allow additional disk space for the error logs.
- Configuring Net.Data:

  If you plan to control the error logging for the whole Net.Data system, set one configuration variable in your Net.Data initialization file: DTW_LOG_DIR

  This variable is required for error logging, even if you have set the DTW_LOG_LEVEL variable in your macro to error or warning. See "DTW_LOG_DIR: Error Log Location Variable" on page 11 to learn how to update the initialization file.
- Writing Net.Data macros:

  Set the level of logging with the DTW_LOG_LEVEL keyword in your macro file.
- Running Net.Data:

  If you are using error logging, then you can check the error logs and archive files for errors in your Net.Data system.
- Tuning:

  Be aware that logging can affect performance. See "Net.Data Error Logging: Performance Considerations" on page 134 for information about performance issues.

Controlling the Amount of Logging in the Log Files

You can specify the level of logging with the DTW_LOG_LEVEL keyword. Define this keyword in the Net.Data macro file. The keyword has three settings:

off Net.Data does not log errors. This is the default.

error Net.Data logs error messages.

warning
Net.Data logs warnings, as well as error messages.

Types of Messages Not Logged

Net.Data does not log errors explicitly handled by a MESSAGE section in the macro file.

Log File Size and Rotation

The maximum size of the log file can be 500 KB. At this size, approximately 3000 log entries will fit.

When the log file reaches the maximum size, the file will be archived to netdata.log*MMMDDYYYY_nn*

Where:

MMM The month (Jan-Dec)

DD The date

YYYY The year

nn A number from 01 to 99, that uniquely identifies each archive file for a certain day.

Logging continues in the original file.

Log File Format

Log file entries have the following format:

```
[DD/MMM/YYYY:HH:MM:SS] [MACRO] [BLOCK] [PID#] [TID#]error_message
```

Parameters:

DD The date

MMM The month (Jan-Dec)

YYYY The year

HH The hour (00-23)

MM The minute (00-59)

SS The number of seconds (00-59)

MACRO
The macro file that generated the error message

BLOCK
The name of the HTML block that generated the error message.

PID# The process ID number of the process that generated the error message. This ID is necessary because multiple Net.Data processes can be writing to the log file.

TID# The thread ID number of the thread that generated the error message. This ID is necessary because multiple threads from the same Net.Data process can be writing to the log file.

error_message
The text of the error message

Appendix A. Net.Data for AIX

Details for AIX are included in the README file that is shipped with Net.Data. The README file includes the following information:

- Requirements
- Installing
- Configuring
- Uninstalling

Loading Shared Libraries for Language Environments

When creating a language environment on the AIX platform, you need to load shared libraries. On AIX, the language environment is required to provide a routine that is called by Net.Data and returns the addresses of the language environment interface routines such as dtw_initialize() and dtw_execute().

Net.Data uses the dtw_fp structure to retrieve pointers to the language environment interface routines from a language environment in AIX, and has this format:

```
typedef struct dtw_fp {
    int (* dtw_initialize_fp)();   /* dtw_initialize function pointer */
    int (* dtw_execute_fp)();      /* dtw_execute function pointer    */
    int (* dtw_getNextRow_fp)();   /* dtw_getNextRow function pointer */
    int (* dtw_cleanup_fp)();      /* dtw_cleanup function pointer    */
} dtw_fp_t;
```

This structure is passed to the language environment by Net.Data as a parameter in the dtw_getFp() routine when the shared library is loaded.

The dtw_fp structure is passed as the only parameter. This structure contains a field for each supported interface, and it is the language environment's responsibility to set these fields. If the language environment is providing the specified interface, it sets the field to the function pointer of that interface. If it is not providing the specified interface, it sets the field to NULL. The dtw_getFp() routine in the program template shows a correct implementation of this routine.

In order for Net.Data to get the pointer to this routine when the shared library is loaded, the dtw_getFp routine must be the first entry point specified in the shared library's export file. A sample export file for a library called dtwsampshr.o that supports all available language environment interface routines looks like this:

```
#!dtwsampshr.o
dtw_getFp
dtw_initialize
dtw_execute
dtw_getNextRow
dtw_cleanup
```

Improving Performance in the REXX Environment

If you have many calls to the REXX language environment on your AIX system, consider setting the RXQUEUE_OWNER_PID environment variable to 0. Macros that make many calls to the REXX language environment can easily spawn many processes, swamping system resources.

You can set the environment variable in one of three ways:

- In the macro file by using the DTW_SETENV built-in function:

  ```
  @DTW_rSETENV("RXQUEUE_OWNER_PID", "0")
  ```

- In the AIX system environment file:

  ```
  /etc/environment: RXQUEUE_OWNER_PID = 0
  ```

  This method affects the behavior of REXX for the whole machine.

- In the HTTP Web server environment file; for example, for Domino Go Webserver, insert the following statement:

  ```
  InheritEnv RXQUEUE_OWNER_PID = 0
  ```

  This method affects the behavior of REXX for the Web server.

Appendix B. Net.Data SmartGuides

The Net.Data SmartGuides are designed to provide you with a quick and easy way to create customized Net.Data applications. Simply select a SmartGuide, answer a few questions, and Net.Data creates a customized application for you.

Net.Data provides the following SmartGuides for you to use while learning how to create macros and use Net.Data features:

Drilldown
: This SmartGuide takes your existing database tables and creates a Web-enabled drilldown application that allows you to access different levels of detail of your data. Optionally, the Drilldown SmartGuide can connect to your database to collect your database information. You can customize up to five Web reports. The generated macro file uses primary and foreign key information stored in your database to link your Web reports automatically.

Forum
: This SmartGuide creates a Web-enabled discussion forum for posting, viewing, and replying to messages. You can customize the forum categories and discussion topics. The generated forum application includes a function to search forum postings for keywords. Optionally, you can enable Cookie functions in your macro file so that the forum can “remember” its users.

Stored Procedure
: This SmartGuide connects to your database and retrieves a list of all stored procedures that are registered with your database. Select a stored procedure and the SmartGuide generates a Net.Data macro file for you that calls your stored procedure. You can then modify the generated macro file or integrate it in your exisiting Net.Data applications.

Contacts
: This SmartGuide generates a Web-enabled address book for storing names, addresses, phone numbers, and other important contact information. It comes with a search function that gives you quick access to your contacts. The generated contacts application can include either a brief or a detailed report. Additionally, you can include a customized report.

Check the Net.Data Web site at `http://www.software.ibm.com/data/net.data` for the latest version of the Net.Data SmartGuide package.

Before You Begin

To run the SmartGuides and generate the Net.Data macro files, you must have the following software installed:

- Java Development Kit (JDK) or Java Runtime Environment (JRE) 1.1.x
- Net.Data Version 2 or higher
- IBM Universal Database (UDB) 5.0 or higher
- REXX (required for the Drilldown SmartGuide)

Running the SmartGuides

The Net.Data SmartGuides are started from the command line and contained in the file, `NetDataSmartGuides.jar`.

To start a SmartGuide with the Java Development Kit (JDK):

1. Add the following line to your CLASSPATH environment variable.

   [Path]`NetDataSmartGuides.jar`

   where *[Path]* is the optional path to the `NetDataSmartGuides.jar` file.
2. If you want to use the database connect feature of the Drilldown and Stored Procedure SmartGuide, add the UDB JDBC driver to your CLASSPATH environment variable.

   For Windows NT and OS/2 operating systems, add the following line to your CLASSPATH environment vairable:

   [Path]`NetDataSmartGuides.jar;`*[UDBInstallationPath]*`\java\db2java.zip`

   For UNIX operating systems, add the following line to your CLASSPATH environment vairable:

   [Path]`NetDataSmartGuides.jar:`*[UDBInstallationPath]*`\java\db2java.zip`

   Where *[Path]* is the optional path to the NetDataSmartGuides.jar file and *[UDBInstallationPath]* is the path to your UDB installation, for example C:\SQLLIB.
3. Start the SmartGuide.
   - For UNIX operating systems, type the following command:

     `java TaskGuide LaunchPad.class`
   - For Windows NT and OS/2 operating systems, run the following file:

     `SmartGuides.cmd`

   A launchpad opens with the list of available SmartGuides as shown in Figure 26 on page 143.

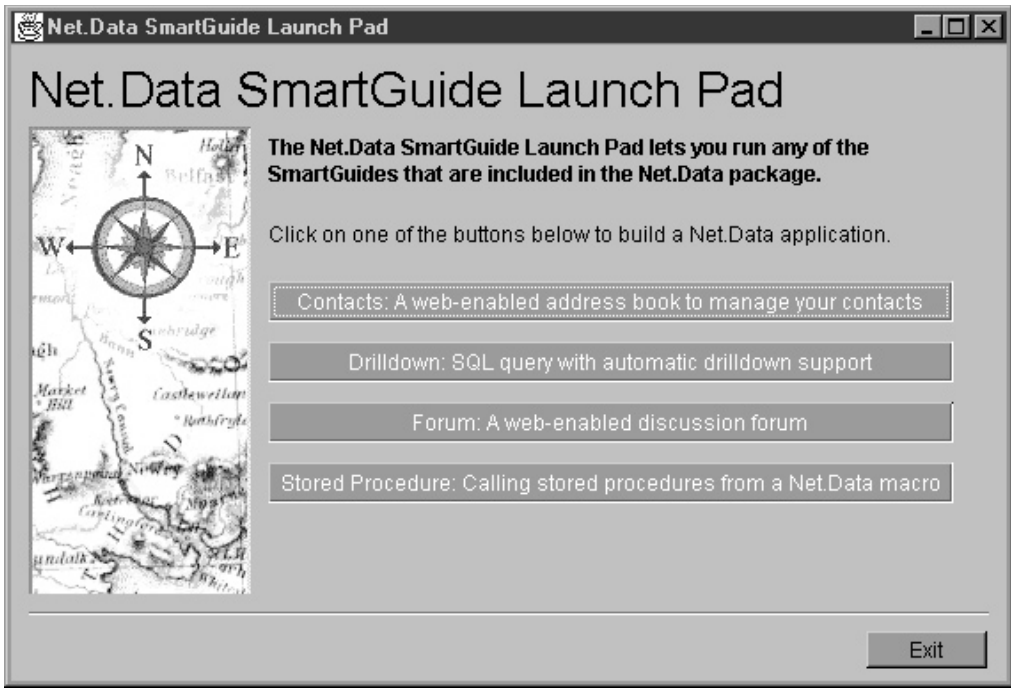


Figure 26. The SmartGuide Lauchpad

4. Click on the name of the SmartGuide that you want to run.

To start a SmartGuide with the Java Runtime Environment (JRE):

1. Type the following command to run the SmartGuides:

   ```
   jre -cp [Path]NetDataSmartGuides.jar TaskGuide LaunchPad.class
   ```

   where *[Path]* is the optional path to the `NetDataSmartGuides.jar` file.

   A launchpad opens with the list of available SmartGuides as shown in Figure 26.

2. If you want to use the database connect feature of the Drilldown and Stored Procedure SmartGuide, type the following command:

   For Windows NT and OS/2 operating systems:

   ```
   jre -cp [Path]NetDataSmartGuides.jar;[UDBInstallationPath]
     \java\db2java.zip TaskGuide LaunchPad.class
   ```

   For UNIX operating systems:

   ```
   jre -cp [Path]NetDataSmartGuides.jar:[UDBInstallationPath]
     \java\db2java.zip TaskGuide LaunchPad.class
   ```

   Where *[Path]* is the optional path to the NetDataSmartGuides.jar file and *[UDBInstallationPath]* is the path to your UDB installation, for example C:\SQLLIB.

3. Click on the name of the SmartGuide that you want to run.

Appendix C. Using NetObjects Fusion NOF Plug-ins with Net.Data Servlets

Net.Data provides a NetObjects Fusion plug-in for the Net.Data servlets. Net.Data servlets are described in “Net.Data Servlets” on page 97.

You can use NetObjects Fusion (NOF) to integrate your existing Net.Data macro files, which will provide better integration with your Web site management and an easy-to-use graphical user interface.

About the NetObjects Fusion Plug-in

The NetDataServlet.NFX plug-in works with the Net.Data servlets. This NOF plug-in supports the invocation of an existing Net.Data macro file or of a single Net.Data function. The plug-in generates the HTML to invoke Net.Data as a servlet or a server-side-include (SSI). When the Web server invokes Net.Data, the Net.Data macro file or function is run. Use the Net.Data servlet plug-in to imbed either of the macro and function servlets into an NOF-managed Web site, which Figure 27 describes. The plug-in provides many of the basic macro file or function parameters, with defaults selected to automate building a macro file. To use the plug-in, you must install and configure NOF and the plug-in files.

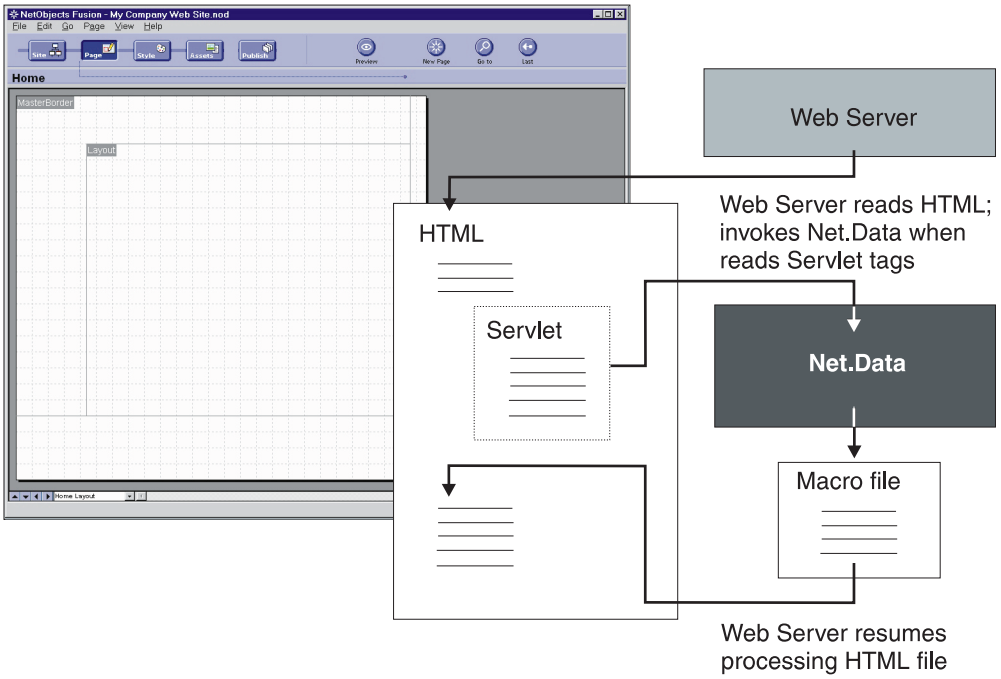


Figure 27. Net.Data Servlets Plug-ins

Installing the NetObjects Fusion Plug-in

Software and Hardware Requirements:

The NOF plug-ins require NetObjects Fusion Version 2.0 or later.

To Install: Copy the <inst_dir>\fx\NetDataServlet.nfx and <inst_dir>\fx\NetDataServlet.gif to your <NetObjects Fusion>\components\ directory.

Setting Up the Net.Data Plug-in for NetObjects Fusion

You can change the properties of the servlet with which you are working using NOF.

1. Open NetObjects Fusion.
2. From the NetObjects Fusion's (NOF) **Tools** palette, select the **NetObjects Components** button:  The plug-in buttons are displayed on the bottom of the **Tools** palette.
3. From these six **Tools** palette buttons, select the **NetObjects Components** button: 
4. On the NOF canvas, mark the area to specify where you want to place the selected plug-in. This is where the results of the servlet will be displayed. The Installed Components window opens, displaying a list of plug-ins from which to choose. If the servlet plug-in is not in the list, use the path and file name fields to specify the plug-in file name: `NetDataServlet.NFX` to use with the macro or function servlet
5. Select the servlet plug-in from the list and click on the **OK** push button.

   The plug-in becomes an object on the NOF canvas.

Modifying the Plug-in Properties

You can modify the macro and function servlets using the Net.Data servlet plug-in.

To modify the Net.Data servlet with NOF:

1. On the NOF canvas, mark the area to specify where you want to place the selected plug-in. This is where the results of the servlet will be displayed. The Installed Components window opens, displaying a list of plug-ins from which to choose. If the servlet plug-in is not in the list, use the path and file name fields to specify the plug-in file name, `NetDataServlet.NFX` to use with the macro or function servlet.
2. Select the Net.Data servlet plug-in from the list and click on the **OK** push button.

   The plug-in becomes an object on the NOF canvas.
3. Select and customize the properties of the Net.Data servlet plug-in:
   a. Select the Net.Data servlet plug-in on the NOF canvas. The NOF **Properties** palette opens, displaying the plug-in properties as shown in Figure 28 on page 147.

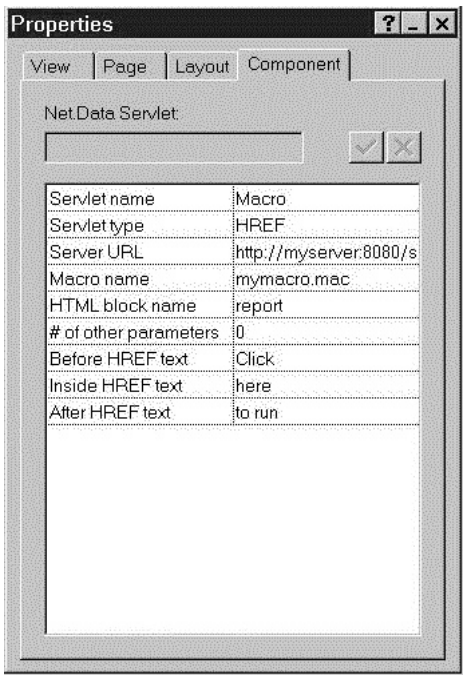


Figure 28. The Net.Data Servlet Properties Palette

Properties for the Net.Data servlets:

You can customize the following properties:

Servlet name
Select the name of the servlet you want to invoke: Function or Macro. Depending on which servlet name you select, different properties are displayed.

Servlet type
Select the type of servlet you want: SSI, HREF, or FORM Submit Button. Depending on which type of servlet you want, different properties are displayed.

Submit label
If you select a FORM Submit Button type, specify the text for the submit label. Otherwise, this property is not displayed.

Server URL
If you select an HREF servlet type, specify the server URL to the servlet-enabled Web server. If you select SSI, this property is not displayed.

Macro name
If you select the macro servlet name, specify the name of a existing Net.Data macro file to execute. Otherwise, this property is not displayed.

HTML block name
If you select the macro servlet name, specify the name of HTML block in the Net.Data macro file to run. Otherwise, this property is not displayed.

Function type
If you select the function servlet name, select the type of function to execute: Function or SQL. Otherwise, this property is not displayed.

Language env
If you select the function servlet name, specify the Net.Data language environment to use. Otherwise, this property is not displayed.

Statement
If you select the function servlet name, specify the statement to execute. Otherwise, this property is not displayed.

Database
If you select the function servlet name, specify the name of the database to use. Otherwise, this property is not displayed.

# of other parameters
Specify the number of other parameters to pass to Net.Data (maximum: 25). For each parameter, enter the name of the parameter and its value (optional).

Before HREF text
If you select an HREF servlet type, specify the text to appear before the text to appear before the `<A HREF>` HTML tag. Otherwise, this property is not displayed (optional).

Inside HREF text
If you select an HREF servlet type, specify the text to appear inside of the `<A HREF>` HTML tag. Otherwise, this property is not displayed (optional).

After HREF text
If you select an HREF servlet type, specify the text to appear after the text to appear after the `<A HREF>` HTML tag. Otherwise, this property is not displayed (optional).

SQL Reminder!
If you select an HREF servlet type and specify an SQL function type, a message displays with a reminder that the HREF SQL statement should use a plus (+) character for any space ( ) characters. This text cannot be changed, nor is it displayed after the page is published. Otherwise, this property is not displayed.

b. After you have defined the properties for your page, click on the **Publish** push button to build and publish the Web pages with the Net.Data servlet NOF plug-in.

Note: If you select a SSI servlet type, your Web page file extension should be `.shtml`. You can set this as your page default from the NOF **Properties** palette, selecting the **Page** notebook tab, clicking on the **Custom names** push button, and entering `.shtml` in the **Extension Type** field.

Publishing Servlets with the NOF Plug-in

After you have set the properties for your page, click on the **Publish** push button to build and publish the Web pages with the plug-in.

Appendix D. Net.Data Sample Macro

This sample macro application displays a list of employees names from which the application user can obtain additional information about an individual employee by selecting the employee's name from the list. The macro uses the SQL language environment to query the EMPLOYEE table for both the employee names and the information about a specific employee.

```
%{************************ Sample Macro ****************************
*   FileName = sqlsamp1.d2w                                          *
*   Description:                                                     *
*     This Net.Data macro file queries...                            *
*         - The EMPLOYEE table to create a selection list of         *
*          employees for display at a browser                        *
*         - The EMPLOYEE table to obtain additional information      *
*          about an individual employee                              *
*                                                                    *
**********************************************************************%}
%{*****************************************************************************
*   Include for global DEFINEs -                                                *
*****************************************************************************%}
%INCLUDE "sqlsamp1.hti"                                                        *
%{*****************************************************************************
*  Function:  queryDB             Language Environment: SQL                   *
*  Description: Queries the table designated by the variable myTable and      *
*  creates a selection list from the result.  The value of the variable       *
*  myTable is specified in the include file sqlsamp1.hti.                     *
*****************************************************************************%}
%FUNCTION(DTW_SQL) queryDB() {
 SELECT * FROM $(myTable)
%MESSAGE {
    -204:  {<p><b>ERROR -204: Table $(myTable) not found. </b>
           <p>Be sure the correct include file is being used.</b>
           %} : exit
    +default: "WARNING $(RETURN_CODE)" : continue
    -default: "Unexpected ERROR $(RETURN_CODE)" : exit
%}

%REPORT {
<select name=emp_name>
%ROW{
<option>$(V2)
%}
</select>
%}
%}

%{*****************************************************************************
*  Function: fname         Language Environment: SQL                          *
*  Description: Queries the table designated by the variable myTable for      *
*               additional information about the employee identified by the   *
*               variable emp_name.                                            *
*****************************************************************************%}
%FUNCTION(DTW_SQL) fname(){
SELECT EMPNME, PHONENO, JOB FROM $(myTable) WHERE EMPNME='$(emp_name)'
%MESSAGE {
  -204: "Error -204: Table not found "
  -104: "Error -104: Syntax error"
   100: "Warning 100: No records" : continue
  +default: "Warning $(RETURN_CODE)" : continue
  -default: "Unexpected SQL error" : exit
%}
%}
```

```
%{*********************************************************************
*  HTML block: INPUT            Title: Dynamic Query Selection          *
*                                                                       *
*  Description: Queries the EMPLOYEE table to create a selection list of    *
*               the employees for display at the browser                *
*********************************************************************%}
%HTML(INPUT) {
<html>
<head>
<title>Generate Employee Selection List</title>
</head>
<body>
<h3>$(exampleTitle)</h3>
<p>This example queries a table and uses the result to create
a selection list using a <em>%REPORT</em> block.
<hr>
<form method="post" action="report">
@queryDB()<input type="submit" value="Select Employee">
</form>
<hr>
</body>
</html>
%}



%{*********************************************************************
*  HTML block:    REPORT                                                *
*  Description: Queries the EMPLOYEE table to obtain additional information *
*               about an individual employee                            *
*********************************************************************%}
%HTML(REPORT) {
<html>
<head>
<title>Obtain Employee Information</title>
</head>
<body>
<h3>You selected employee name = $(emp_name)</h3>
<p>Here is the information for that employee:
<PRE>
@fname()
</PRE>
<hr><a href="input">Return to previous page</a>
</body>
</html>
%}

%{     End of Net.Data macro 1 %}
==========================================================================
%{************************** Include File ********************************
*   FileName = sqlsamp1.hti                                             *
*   Description:                                                        *
*     This include file provides global DEFINEs for the sqlsamp1.d2w        *
*     Net.Data macro.                                                   *
*********************************************************************%}
%define {
    emp_name   =""
    reposition = sign
 exampleTitle = "Sample Macro"
    myTable = "MRZ.EMPLOYEE"
%}

%{     End of include file  %}
```

Appendix E. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this publication 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 as your own risk.

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

_W92/H3_
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
_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.

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.

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.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
AS/400
CBIDO
CBPDO
CICS
CustomPac
DB2
DB2 Universal Database
DataJoiner
Distributed Relational Database Architecture
DRDA
IBM
IMS
Language Environment
MVS/ESA
Net.Data
OpenEdition
OS/2
OS/390
OS/400
RACF
SystemPac

The following terms are trademarks of other companies as follows:

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Lotus and Domino Go Webserver are trademarks of Lotus Development Corporation in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Glossary

API. Application programming interface.

applet. A Java program included in an HTML page. Applets work with Java-enabled browsers, such as Netscape, and are loaded when the HTML page is loaded.

application programming interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program. Net.Data supports the following proprietary Web server APIs for improved performance over CGI processes: ICAPI, GWAPI, ISAPI, and NSAPI.

BLOB. Binary large object.

cache. A part of memory or disk space that contains recently accessed data, designed to speed up subsequent access to the same data. The cache is often used to hold a local copy of frequently-used data that is accessible over a network.

caching. The processes of storing frequently-used results from a request to the Web server locally for quick retrieval, until it is time to refresh the information.

Cache Manager. The program that manages a cache for one machine. It can manage multiple caches.

CGI. Common Gateway Interface.

cliette. A long-running process in Net.Data Live Connection that serves requests from the Web server. The Connection Manager schedules cliette processes to serve these requests.

CLOB. Character large object.

Common Gateway Interface. A standardized way for a Web server to pass control to an application program and receive data back.

Connection Manager. An executable file, `dtwcm`, in Net.Data that is needed to support Live Connection.

cookie. A packet of information sent by an HTTP server to a Web browser and then sent back by the browser each time it accesses that server. Cookies can contain any arbitrary information the server chooses and are used to maintain state between otherwise stateless HTTP transactions. *Free Online Dictionary of Computing*

database. A collection of tables, or a collection of table spaces and index spaces.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

data type. An attribute of columns and literals.

DBMS. Database management system.

firewall. A computer with software that guards an internal network from unauthorized external access.

flat file interface. A set of Net.Data built-in functions that let you read and write data from plain-text files.

HTML. Hypertext markup language.

HTTP. Hypertext transfer protocol.

hypertext markup language. A tag language used to write Web documents.

hypertext transfer protocol. The communication protocol used between a Web server and browser.

ICAPI. Internet Connection API.

ICS. Internet Connection Server.

ICSS. Internet Connection Secure Server.

Internet. An international public TCP/IP computer network.

Internet Connection Server. IBM's unsecure Web server.

Internet Connection Secure Server. IBM's secure Web server.

Intranet. A TCP/IP network inside a company firewall.

ISAPI. Microsoft's Internet Server API.

Java. An operating system-independent object-oriented programming language especially useful for Internet applications.

language environment. A module that provides access from a Net.Data macro to an external data source such as DB2 or a programming language such as Perl. Some language environments are supplied with Net.Data such as REXX, Perl, and Oracle. You can also create your own language environments.

Live Connection. A Net.Data component that consists of a Connection Manager and multiple cliettes. Live Connection manages the reuse of database and Java virtual machine connections.

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.

NSAPI. Netscape API.

null. A special value that indicates the absence of information.

path. A search route used to locate files.

Perl. An interpreted programming language.

port. A 16-bit number used to communicate between TCP/IP and a higher-level protocol or application.

TCP/IP. Transmission Control Protocol / Internet Protocol.

Transmission Control Protocol / Internet Protocol. A set of communication protocols that support peer-to-peer connectivity functions for both local and wide-area networks.

URL. Uniform resource locator.

uniform resource locator. An address that names a HTTP server and optionally a directory and file name, for example:
`http://www.software.ibm.com/data/net.data/index.html`.

Web server. (1) A computer running HTTPmiddleware (2) server software, such as Internet Connection.

Index

D

E

F

G

H

I

O

P

R

S

T

U

V

W

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.