Net.Data

Language Environment Interface Reference

Net.Data

Language Environment Interface Reference

Note

Be sure to read the information in “Appendix D. Notices” on page 59 before using this information and the product it supports.

Sixth Edition (May 1999)

Contents

Preface

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

About Net.Data

With IBM's Net.Data product, you can create dynamic Web pages using data from both relational and non-relational database management systems (DBMSs), including DB2, IMS, ODBC-enabled databases, and databases that can be accessed through DRDA, and using applications written in programming languages such as Java, JavaScript, Perl, C, C++, and REXX. The Net.Data family of products provides similar capabilities on machines executing the Windows NT, AIX, OS/2, OS/390, OS/400, HP-UX, Sun Solaris, Santa Cruz Operating System (SCO), and Linux operating systems.

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

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

- Encapsulate business logic within applications written in, but not limited to, C, C++, RPG, COBOL, JAVA, Perl, or REXX programming languages
- Access databases such as DB2
- Access other data sources such as flat files

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

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

For improved performance, Net.Data supports a variety of Web server Application Programming Interfaces (APIs). In addition, Net.Data can be lauched as a Java servlet.

About This Book

This book describes Net.Data's Language Environment Interface (LEI), which you can use to develop your own custom language environments for Net.Data.

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

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

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

Who Should Read This Book

People who want to extend the functionality of Net.Data to meet the needs of their particular enterprise can use this book to write their own language environments for Net.Data.

To understand the concepts discussed in this book, you should be familiar with the following information:

- The C programming language
- The information in *Net.Data Administration and Programming Guide* and *Net.Data Reference*

About Examples in This Book

Examples used in this book are kept simple to illustrate specific concepts and do not consider every possible case. Some examples are fragments that do not work alone.

About Net.Data Language Environments

Net.Data is designed to allow new programming language and database interfaces to be added in a pluggable fashion. These interfaces are called language environments and are accessed as DLLs or shared libraries. Language environments provide access to applications and databases that support your dynamic Web pages. By invoking language environments with function calls, you can use the functionality that these language environments provide for use with your business application. For example, you can directly access your ODBC database, use the Perl language environment to execute Perl scripts, or call the Java Applets language environment to run Java applets.

The Net.Data initialization file associates each language environment name with a DLL or shared library. Each language environment must support a standard set of interfaces defined by Net.Data. Net.Data loads the DLL or shared library specified in the initialization file the first time that a function call for a FUNCTION block specifying that language environment is encountered.

Net.Data parses the Net.Data macro, maintains the Net.Data variables, communicates with the language environments, and formats the output according to the REPORT and MESSAGE block specifications. The language environment supports the interfaces defined to Net.Data, makes the Net.Data parameters accessible to the language processor in some language-dependent manner, calls the language interpreter, and receives the variables back from the language interpreter in some language-dependent manner.

Figure 1 demonstrates Net.Data's interaction with language environments.

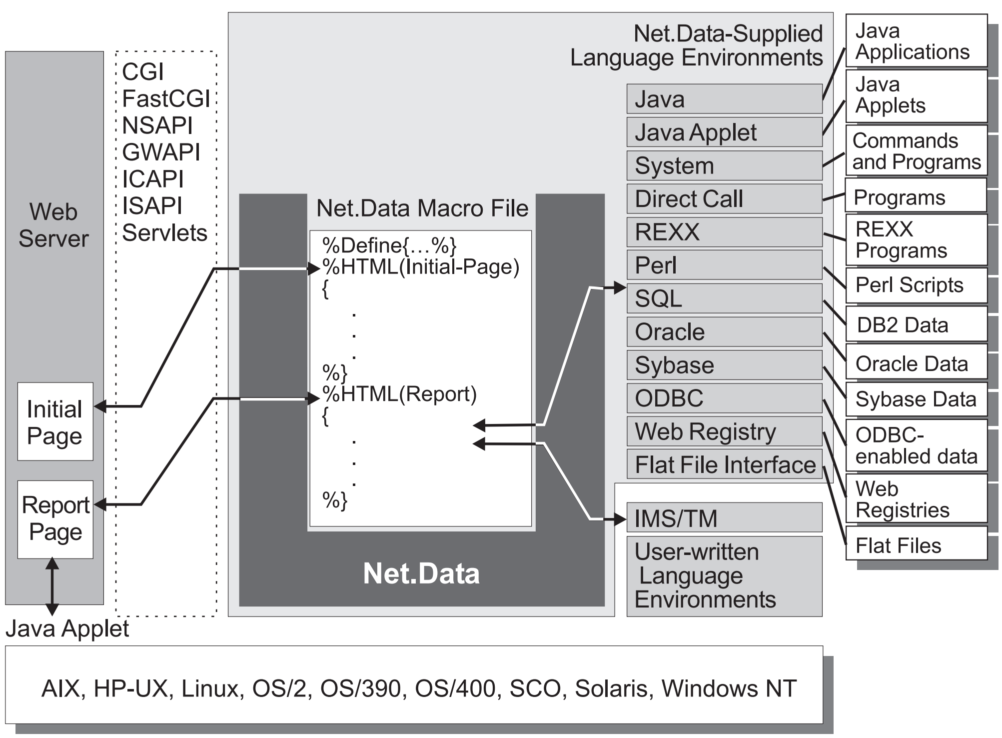

Figure 1. Net.Data and Language Environments

This book describes the Net.Data language environment interface used to create new language environments. The Net.Data-supplied language environments are described in the language environment chapter of *Net.Data Administration and Programming Guide* for your operating system.

Chapter 1. Creating a New Language Environment

Net.Data uses language environments as pluggable programming language and database interfaces, accessed as DLL files, shared libraries, or service programs depending upon your operating system environment. In this document the term *shared library* is used when generically referring to these types files. Net.Data provides a set of language environments, but when these language environments do not meet your application needs, you can create your own, using the Net.Data language environment interface.

Creating a new language environment involves the following steps:

- Determine what interfaces and functions you must provide for the language environment. The dtw_execute() interface must be provided, and all provided interfaces must match exactly the prototypes that are defined in the dtwle.h C language header.
- Build a shared library that implements the set of language environment interface routines you want to provide. See the documentation for your compiler to understand how to build shared libraries.
- Make all interfaces externally available from the shared library so Net.Data can call them.
- Determine your ENVIRONMENT configuration statement, then add it to the Net.Data initialization file.
- Add functions to the Net.Data macro that uses the new language environment.

Before you decide to create a new language environment, determine if the Net.Data-supplied language environments satisfy your requirements.

This chapter describes how to design the language environment.

- "Creating a Shared Library"
- "Language Environment Communication Structures" on page 3
- "Language Environment Interface Functions" on page 6
- "Designing the Language Environment Statement" on page 9

To learn about the language environment programming interface, see "Chapter 2. The Language Environment Programming Interface Utility Functions" on page 13.

Creating a Shared Library

When you build a language environment, you can use the template supplied in "Appendix B. Language Environment Template" on page 41, which provides the environment interface functions and the communication structures used by Net.Data to communicate with your language environment and to pass parameters to and from the language environment.

The following sections describe concepts and design issues for the functions and structures. The utilities provided in the language environment interface are described in "Chapter 2. The Language Environment Programming Interface Utility Functions" on page 13.

- "Which Language Environment Interfaces Should I Provide?" on page 2
- "Processing Input Parameters" on page 2
- "Processing User Requests" on page 3

- “Processing Output Parameters” on page 3
- “Communicating Error Conditions” on page 3

Which Language Environment Interfaces Should I Provide?

When you write a language environment, you must determine which interfaces to provide. Your choices depend on what you intend the language environment to do. For example, if the language environment will be accessing database data, you’ll make different choices than if it is for a scripting language. The following section describes the Net.Data language environment interfaces.

dtw_execute()
: You must provide the dtw_execute() interface to pass input parameters from the macro; it is the only required interface for every language environment. Net.Data passes all input parameters to dtw_execute() through the language environment communication structure, dtw_lei_t .

dtw_initialize()
: Provide the dtw_initialize() interface to allocate or initialize data. Net.Data calls this interface only once for each macro invocation, before the first function call to your language environment. If there are no function calls to your language environment, Net.Data does not call the dtw_initialize() interface.

dtw_cleanup()
: Provide the dtw_cleanup() interface when you provide a dtw_initialize() interface, and you want to release any resources.

dtw_getNextRow()
: Provide the dtw_getNextRow() interface as part of a database language environment or a language environment that can process data a row at a time. This interface is called if Net.Data is running on the OS/400® or OS/390 operating systems.

Processing Input Parameters

The Net.Data language environments use the dtw_execute() interface to receive and process parameters. The dtw_execute() interface works with the dtw_lei_t structure, which is used by Net.Data to communicate with the language environment. Use the following recommendations for input parameter processing, when writing your language environment.

- Specify any implicit parameters in the ENVIRONMENT statement for the language environment, in the Net.Data initialization file. Net.Data passes the parameters specified here on all function calls to the language environment after it passes the parameters specified by the macro writer on the FUNCTION block being executed.
- Receive input parameters to the dtw_execute() interface as part of the dtw_lei_t structure. The macro writer determines the order that Net.Data passes the parameters when specifying them in the FUNCTION block definition of the Net.Data macro.

The processInputParms() routine in the program template, in “Appendix B. Language Environment Template” on page 41 shows one method of processing input parameters.

Processing User Requests

How a language environment processes a user request depends on how the language environment receives the request. Net.Data provides several different ways for you to communicate a request to your language environment:

- Through the function name specified on a FUNCTION block. On every function call, Net.Data passes the function name to the language environment in the function_name field of the dtw_lei_t structure.
- Through the FUNCTION block parameter list. You can specify that a parameter in the parameter list can indicate a user request. On every function call, Net.Data passes parameters to the language environment in the parm_data_array field of the dtw_lei_t structure.
- Through the executable-statements section of a FUNCTION block. On every function call, Net.Data passes any executable statements specified in the FUNCTION block to the language environment in the exec_statement field of the dtw_lei_t structure.

Processing Output Parameters

The method you use to process output parameters depends entirely on your language environment and how it processes user requests. However, once the language environment has the data it needs to return to the Net.Data macro, you can design the language environment to modify the values of parameters passed in the parm_data_array field of the dtw_lei_t structure. The processOutputParms() routine in the program template, in “Appendix B. Language Environment Template” on page 41, shows one possible way of processing output parameters, as well as examples of how to set both string and table parameter values.

Communicating Error Conditions

The success or failure of a function call can be communicated through the implicit Net.Data macro variable, RETURN_CODE. This variable is set by Net.Data after returning from a call to the dtw_execute() interface. Its value is set to the return value of the dtw_execute() call itself. This value is then used by Net.Data to process the Net.Data macro MESSAGE block, if one was specified for this function call.

If you do not specify a MESSAGE block, or do not have an entry in a specified MESSAGE block to handle the return code from dtw_execute(), Net.Data displays the contents of the default_error_message field of the dtw_lei_t structure. This field can be set by the language environment at any time in the dtw_execute() routine. The setErrorMessage() routine in the program template, in “Appendix B. Language Environment Template” on page 41, shows an example of how to set the default_error_message field.

Language Environment Communication Structures

Net.Data uses two structures to communicate with your language environment. Your language environment must work with these structures and set and pass information within the structures.

- dtw_lei_t
- dtw_parm_data_t

Net.Data passes a language environment interface structure (for example, dtw_lei_t) to the language environment function that it calls. The structure contains, among other things, a parameter data array that contains a list of parameters to be passed to the language environment function. The language environment function called by Net.Data processes the request, updates the parameters in the parameter data array (if applicable), and returns to Net.Data.

Net.Data then goes through the parameter data array, updates its copies of the parameters to reflect the new values set by the language environment function, and continues the processing of the Net.Data macro.

The dtw_lei_t Structure

The interface function of each language environment receives a pointer to the dtw_lei_t structure. The dtw_lei_t structure has the following format:

```
typedef struct dtw_lei_t {              /* Lang. Env. Interface       */
    char *function_name;              /* Function block name        */
    int   flags;                      /* Lang. Env. Interface flags  */

    char *exec_statement;             /* Lang. Env. statement(s)     */

    dtw_parm_data_t *parm_data_array; /* Parameter array             */
    char *default_error_message;      /* Default message             */
    void *le_opaque_data;             /* Lang. Env. specific data    */

    void *row;                        /* For row-at-a-time processing*/

    char  reserved[64];               /* Reserved                    */
} dtw_lei_t;
```

Fields in the dtw_lei_t structure:

function_name
The function_name field contains a pointer to a string containing the name of the function block. This can be useful to specify the FUNCTION block name in error messages displayed by the language environment.

flags The flags field is used by Net.Data to communicate with the language environment. Specify the flags field pointer by performing an OR operation using the following constants:

- Net.Data sets DTW_STMT_EXEC to tell the dtw_execute() interface function that the exec_statement field contains the file name and parameters from an EXEC statement.
- DTW_END_ABNORMAL is set by Net.Data to tell the dtw_cleanup() interface function that an abnormal or unexpected condition has occurred and that the language environment should perform any cleanup necessary (that is, free held resources) before Net.Data ends.
- DTW_LE_FATAL_ERROR is set by a language environment interface function to tell Net.Data that a fatal error has occurred in the language environment. If this flag is set, Net.Data stops processing the Net.Data macro, calls all active language environment's dtw_cleanup() interface function with flags set to DTW_END_ABNORMAL, prints default message, and exits. The flag is checked only if a non-zero return value is returned on a language environment call.
- DTW_LE_MSG_KEEP is set by a language environment interface function to tell Net.Data that the storage pointed to by default_error_message should not be freed. If this constant is not set, Net.Data attempts to free the storage.

- DTW_LE_CONTINUE is set by the dtw_execute() interface function to tell Net.Data to call the dtw_getNextRow() interface function. Net.Data calls dtw_getNextRow() only if the flag is set and the return value from the call to the dtw_execute() interface function is zero.

exec_statement
The exec_statement field contains one of the following pointers:
- To a string containing the executable statements (after variable substitution) from the FUNCTION block
- To the file name and parameters from an EXEC statement

parm_data_array
The parm_data_array field contains a pointer to an array of dtw_parm_data_t structures. The array ends with a parm_data structure containing zeros. The dtw_parm_data_t structure is used by Net.Data to pass variables and the associated value to a language environment and to retrieve any changes to the variable value that may be made by the language environment. See “The dtw_parm_data_t Structure” for a description of the structure.

default_error_message
The default_error_message field is set by the language environment to a character string that describes an error condition. If the return value from a call to a language environment interface function is non-zero and the return value does not match the value of a message in a MESSAGE block, the default message is displayed. Otherwise, Net.Data displays the message selected from the MESSAGE block.

le_opaque_data
The le_opaque_data field is set by any of the interface functions in the language environment to pass parameters from one interface function to another. Net.Data saves the pointer and passes it to another interface function that Net.Data calls. After processing the Net.Data macro, and before returning to the caller of Net.Data, Net.Data defines the pointer to NULL. Because the field is thread-specific, language environments can store data that is thread specific. Use this field only if you have a dtw_cleanup() interface function, so that the function can free the storage associated with the le_opaque_data field.

row The row field is set by Net.Data to a row object prior to calling a language environment’s dtw_getNextRow() interface function. The dtw_getNextRow() function inserts a row of table data in the object using the Net.Data row utility interface functions. Net.Data then processes the row and calls dtw_getNextRow() until there are no more rows to process.

The `reserved` field is for IBM use only.

The dtw_parm_data_t Structure

Net.Data uses the dtw_parm_data_t structure to pass parameters to a language environment. Parameters are obtained from three sources:
- Explicit parameters that are specified on the FUNCTION block definition
- The return variable that is specified on the RETURNS keyword on a FUNCTION block definition
- Parameters that are specified on the ENVIRONMENT configuration statement in the Net.Data initialization file

Net.Data passes explicit parameters first, followed by parameters specified in the ENVIRONMENT statement, and then the return variable.

The dtw_parm_data_t structure has the following format:

```
typedef struct dtw_parm_data_t {        /* Parameter data              */
    int   parm_descriptor;            /* Parameter descriptor        */
    char *parm_name;                  /* Parameter name              */
    char *parm_value;                 /* Parameter value             */
    void *res1;                       /* Reserved                    */
    void *res2;                       /* Reserved                    */
} dtw_parm_data_t;
```

Fields in the dtw_parm_data_t structure:

parm_descriptor
: The parm_descriptor field describes the type and use of the parameter being passed to the language environment. Net.Data sets the field by performing an OR operation using the following constants:

- DTW_IN indicates that a parameter is an input-only parameter.
- DTW_OUT indicates that a parameter is an output-only parameter.
- DTW_INOUT indicates that a parameter is an input and output parameter.
- DTW_STRING indicates that parameter value is a pointer to a string.
- DTW_TABLE indicates that the parameter value is a pointer to a table.

Net.Data always sets the parm_descriptor field to DTW_IN, DTW_OUT, or DTW_INOUT and uses a logical OR with DTW_STRING and DTW_TABLE.

parm_name
: The parm_name field is a pointer to a string that contains the name of the parameter. Net.Data sets this pointer NULL if the parameter is a literal string.

parm_value
: The parm_value field is a pointer to an object that contains the value of the parameter. This pointer is set to NULL by Net.Data if the parameter is a variable that is not already defined.

The res1 and res2 fields are reserved fields.

Both parm_name and parm_value point to an object allocated from the Net.Data run-time *heap*, the area of memory used for dynamic memory allocation by Net.Data. If parm_name or parm_value is replaced with another string, the original string must be freed and replaced with a pointer to a character string allocated from the Net.Data heap. Use the dtw_malloc() and dtw_free() utility functions to free the original string.

Language Environment Interface Functions

Net.Data uses four interface functions with a language environment: you provide one or more of these functions. Three of these functions are optional, but every language environment must have a dtw_execute() interface function. If a Net.Data macro references a language environment that does not have a dtw_execute() interface function, Net.Data returns an error message and stops processing the Net.Data macro.

To call a language environment, reference it on the FUNCTION block of the Net.Data macro. The language environment interface functions must be called in the following order:
1. dtw_initialize()
2. dtw_execute()
3. dtw_getNextRow()
4. dtw_cleanup()

The dtw_execute() function is the only interface function that you must provide in the language environment.

When Net.Data encounters a call to a function that uses the language environment, it uses the following steps to call the language environment:
1. Net.Data calls dtw_initialize() if it has been defined for this language environment. The function performs any initialization tasks required by the language environment, such as connecting to databases, or allocating variables.
2. Net.Data calls dtw_execute() to process the macro FUNCTION block containing statements that the language environment must process.
3. Net.Data calls dtw_getNextRow() if, on successful return, dtw_execute() indicated that dtw_getNextRow() should be called.
4. When the Net.Data macro processing is complete, Net.Data calls dtw_cleanup() to clean up the environment (for example, disconnecting from the database or freeing variables) if this function has been defined for the language environment, and then returns to the Web server.

The following sections describe the interface functions:
- "dtw_initialize()"
- "dtw_execute()"
- "dtw_getNextRow()" on page 8
- "dtw_cleanup()" on page 9

dtw_initialize()

The dtw_initialize() interface function performs any special initialization that the language environment requires, such as connecting to a database or allocating variables. This interface function is called once and is optional.

Net.Data calls a language environment's dtw_initialize() interface function only once per macro, the first time Net.Data calls a FUNCTION block referencing that language environment. Subsequent references to the language environment bypass the call to the dtw_initialize() interface function.

This interface function does not affect message block processing. A positive or zero return code means that processing continues; a negative return code means that processing does not continue. If the return code is non-zero and there is a default message defined in the default_error_message field, the default message is issued; if there is no default message, Net.Data issues an error message.

dtw_execute()

The dtw_execute() interface function processes macro FUNCTION blocks that contain statements that must be processed by the language environment. For example, a FUNCTION block that refers to a database language environment contains SQL statements that language environment uses to query the database.

The dtw_execute() interface function is called whenever a Net.Data macro processes a FUNCTION block that refers to the language environment. When the dtw_execute() interface function completes, what happens next depends on whether the language environment is processing table data a row at a time. If so, the interface function sets DTW_LE_CONTINUE flag in the dtw_lei_t structure to tell Net.Data to call the dtw_getNextRow() interface function. See "dtw_getNextRow()" for more information about the dtw_getNextRow() interface function and its processing steps.

You can optimize performance by having the dtw_execute() interface function do all the processing necessary to produce the input for the report block processing. For example, your dtw_execute interface function can generate an entire table to be processed during the report block phase

dtw_getNextRow()

The dtw_getNextRow() interface function retrieves input for row-at-a-time processing of Net.Data tables. It is called each time the DTW_LE_CONTINUE flag is set, indicating that another row of data needs to be processed for the table. Use dtw_getNextRow() for database language environments.

Restriction: This interface function is only called if Net.Data is running on the OS/400 or OS/390 operating systems.

Net.Data calls dtw_getNextRow() when the following conditions are met:

- The call to the language environment's dtw_execute() call completes successfully (return value of zero)
- The dtw_execute() interface function has set the DTW_LE_CONTINUE flag in the dtw_lei_t structure.

When the dtw_execute() function sets the DTW_LE_CONTINUE flag to on, Net.Data performs the following steps:

1. Processes the message block for the return value of the dtw_execute() interface function.
2. Calls language environment's dtw_getNextRow() interface function and begins row-at-a-time processing.
3. Processes the report block.
4. Processes the message block for the return value of the dtw_getNextRow() interface function.
5. Determines whether dtw_getNextRow() has turned on the DTW_LE_CONTINUE flag:
   - If yes, processing continues with the dtw_getNextRow() interface function in step 2.
   - If no, row-at-a-time processing ends and Net.Data continues processing the Net.Data macro.

When dtw_getNextRow() is called, the row field in the dtw_lei_t structure is set to point to a row object. To manipulate the row object, use the Net.Data utility functions, dtw_row_SetCols() and dtw_row_SetV(). Net.Data assumes that after the first call to the dtw_getNextRow() interface function the row object contains the column headings for the table. Subsequent calls contain the actual table data.

The dtw_getNextRow() function continues to get called (unless message block processing indicates otherwise) as long as the DTW_LE_CONTINUE flag is set.

dtw_cleanup()

Use the dtw_cleanup() interface function to cleanup the language environment if you use dtw_initialize() to initialize the language environment. Use this function for such tasks as disconnecting from a database or freeing variables. This interface function is optional.

While handling a Net.Data request, Net.Data calls a language environment's dtw_cleanup() interface function once when either Net.Data processing ends or an error stops Net.Data from processing the macro.

Net.Data sets the flags field in the dtw_lei_t structure to DTW_END_ABNORMAL if the cleanup processing is abnormal. The following abnormal conditions provide examples of when to use dtw_cleanup():

- A language environment interface function indicates that a fatal error occurred by setting the DTW_LE_FATAL_ERROR bit in the flags field in the dtw_lei_t structure.
- Net.Data encounters an unrecoverable error.
- The Net.Data macro message block processing results in an exit.

If a language environment's interface function sets the le_opaque_data field with a parameter to be passed between interface functions, use the dtw_cleanup() to free the field when processing ends.

This interface function does not affect message block processing. If the return value is non-zero, a default message is issued; if no default message exists, the macro processor issues a warning message.

Designing the Language Environment Statement

Each language environment has an ENVIRONMENT statement in the Net.Data initialization file that contains information specific to that language environment. When you create a new language environment, you need to design an environment statement for the initialization file and document how users should add it to the initialization file.

The ENVIRONMENT statements specify information about the language environment that Net.Data requires to call and load the language environment DLL or shared library, such as the language environment name, the DLL or shared library name, and the list of parameters to be passed to the language environment for each function call.

Net.Data reads the configuration information when it is invoked, but does not load language environment DLLs or shared libraries until a FUNCTION block identifying that language environment is called from within the macro. The DLL remains loaded until Net.Data ends.

The following sections provide information about syntax, parameter descriptions, and examples that you can use in your documentation.

ENVIRONMENT Statement Syntax

An ENVIRONMENT statement has the following format:

```
ENVIRONMENT(type) library-name  ([specification parameter_list, ...)
```

Each ENVIRONMENT statement must be on a single line.

The following are the parameters you must specify for each language environment:

- *type*

  The name that associates this language environment with a FUNCTION block definition in a Net.Data macro. You must also specify the language environment type on a FUNCTION block definition to tell Net.Data which language environment processes the function call. See the ″Function Block″ section in *Net.Data Reference* for more information about the FUNCTION block.

  Important: The name cannot begin with the prefix DTW. This prefix is reserved for language environments shipped with Net.Data. If you use the DTW prefix, Net.Data cannot load your language environment DLL.
- *library_name*

  The name of the object containing the language environment interfaces that are called by Net.Data. The file extension is different for each operating system:
  - In AIX®, the name of the shared library is specified with the *.o* extension.
  - In HP/UX, the name of the shared library is specified with the *.sl* extension
  - In OS/2® and Windows NT, the DLL name is specified with the *.dll* extension.
  - In OS/390®, the DLL name is specified without the *.dll* extension.
  - In OS/400®, the service program name is specified with the *.SRVPGM* extension.

    In SUN, SCO, and LINUX the name of the shared library is specified with the *.so* extension

  Look at the initialization file shipped with Net.Data for your operating system to see how to specify this name. Consider using a fully qualified path name to make sure Net.Data finds the DLL or shared library.
- *specification*

  The parameter passing specification that indicates whether Net.Data uses the parameter for input, output, or input and output. Possible values:

IN	An parameter used for input
OUT	A parameter used for output
INOUT	A parameter used for both input and output
- *parameter_list*

  The list of parameters that are passed to the language environment on each function call, in addition to those parameters specified in the FUNCTION block definition. They are passed in the *parm_data_array* field of the *dtw_lei_t* structure following the parameters specified in the FUNCTION block definition. You must define these parameters as variables in your Net.Data macro before the function call is made. If a function modifies the value of these parameters, the parameters retain the modified value once the function finishes processing.

ENVIRONMENT Statement Examples

The following examples show ENVIRONMENT statements for language environments that Net.Data supplies. These examples illustrate how to specify parameters. The variables you include in the ENVIRONMENT statements are ones that you want to allow Net.Data macro writers to define or override in their macros. See the operating system-specific information in the appendixes in *Net.Data Reference* or in your Net.Data README file or Program Directory for additional examples.

The following examples show ENVIRONMENT statements for the Net.Data-provided langauge environment using the syntax for OS/2, AIX, and Windows NT.

```
ENVIRONMENT (DTW_SQL)      DTWSQL    ( IN DATABASE, LOGIN, PASSWORD,
TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_SYB)      DTWSYB    ( IN DATABASE, LOGIN, PASSWORD,
TRANSACTION_SCOPE, SHOWSQL, ALIGN, START_ROW_NUM, DTW_SET_TOTAL_ROWS)
ENVIRONMENT (DTW_ORA)      DTWORA    ( IN 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)   () CLIETTE "DTW_JAVAPPS"
ENVIRONMENT (DTW_PERL)     DTWPERL   ()
ENVIRONMENT (DTW_REXX)     DTWREXX   ()
ENVIRONMENT (DTW_SYSTEM)   DTWSYS    ()
ENVIRONMENT (HWS_LE)       DTWHWS    ()
```

ENVIRONMENT statement can vary on each operating systems; for example OS/390 differs slightly for SQL and ODBC access:

```
ENVIRONMENT (DTW_SQL)      DTWSQL    ( IN LOCATION, DB2SSID, DB2PLAN,
TRANSACTION_SCOPE)

ENVIRONMENT (DTW_ODBC)     DTWODBC   ( IN LOCATION, TRANSACTION_SCOPE)
```

Chapter 2. The Language Environment Programming Interface Utility Functions

Net.Data provides a programming interface for you to use when designing a new language environment. The language environment interface has utility functions that access Net.Data services that manage memory and configuration variables, and provide table and row manipulation features. "Appendix B. Language Environment Template" on page 41 provides a template that you can use as a model when designing your language environment.

The following section explains the Net.Data language environment interface utility functions.

Language Environment Utility Functions

Language environments use utility functions to access Net.Data services. These functions fall into four categories:
- "Utility Functions for Managing Memory"
- "Utility Functions for Managing Configuration Variables"
- "Utility Functions for Table Manipulation" on page 14
- "Utility Functions for Row Manipulation" on page 15

Utility Functions for Managing Memory

Language environments use the memory management utility functions to allocate storage owned by Net.Data, and to free storage that it allocated using the Net.Data run-time library.

The following example illustrates the need for these utility functions. Suppose that Net.Data is written using compiler A, with its corresponding run-time library. A programmer writes a new language environment, but uses compiler B, which has a different run-time library. The language environment cannot free storage that Net.Data allocated, and Net.Data cannot free storage that was allocated by the language environment because of potential incompatibilities between the two run-time libraries.

Table 1. Memory Management Utility Functions

Utility Function	Description
"dtw_malloc()" on page 18	Allocate storage from Net.Data's run-time heap using dtw_malloc().
"dtw_free()" on page 16	Free storage allocated from Net.Data's run-time heap using dtw_malloc().
"dtw_strdup()" on page 21	Allocate storage from Net.Data's run-time heap and copy the specified string into the allocated storage using dtw_malloc().

Utility Functions for Managing Configuration Variables

The management utility functions for the configuration variables let language environments access configuration information stored in the Net.Data initialization file. Using these functions, all language environments can share the Net.Data initialization file and use information in it for configuring language environments.

Table 2. Configuration Utility Functions

Utility Function	Description
"dtw_getvar()" on page 17	Retrieve the value of a configuration variable from the Net.Data initialization file.

Utility Functions for Table Manipulation

Use the table functions to manipulate any Net.Data macro table variables that are passed to the language environment.

Row and column numbers begin with one (1).

Table 3. Table Utility Functions

Utility Function	Description
"dtw_table_New()" on page 32	Create a table object.
"dtw_table_Delete()" on page 24	Delete a table object.
"dtw_table_SetCols()" on page 35	Set the width of a table and allocate storage for the column headers.
"dtw_table_GetV()" on page 28	Retrieve a table value.
"dtw_table_SetV()" on page 37	Set a table value.
"dtw_table_GetN()" on page 27	Retrieve a table column heading.
"dtw_table_SetN()" on page 36	Set a table column heading.
"dtw_table_Rows()" on page 34	Retrieve the current number of rows in a table.
"dtw_table_Cols()" on page 23	Retrieve the current number of columns in a table.
"dtw_table_MaxRows()" on page 31	Retrieve the maximum allowable number of rows in a table.
"dtw_table_QueryColnoNj()" on page 33	Retrieve the column number of a column.
"dtw_table_AppendRow()" on page 22	Add one or more rows to the end of a table.
"dtw_table_InsertRow()" on page 30	Insert one or more rows in a table.
"dtw_table_DeleteRow()" on page 26	Delete one or more rows from a table.
"dtw_table_InsertCol()" on page 29	Insert one or more columns in a table.
"dtw_table_DeleteCol()" on page 25	Delete one or more columns from a table.

Utility Functions for Row Manipulation

The row utility functions manipulate the row object that is passed to a language environment's dtw_getNextRow() interface function during row-at-a-time processing.

Row numbers begin with one (1).

Table 4. Row Utility Functions

Utility Function	Description
"dtw_row_SetCols()" on page 19	Set the width of a row.
"dtw_row_SetV()" on page 20	Set a table value.

Utility Functions Syntax Reference

This section describes each of the utility functions, their format, usage, and parameters, as well as providing a simple example.

dtw_free()

Usage

Frees storage that was allocated from Net.Data's run-time heap using dtw_malloc(). The buffer points to the allocated storage to free.

Format

```
void dtw_free(void *buffer)
```

Parameters

buffer	A pointer to the allocated storage to free.

Examples

```
char *myBuf;
long  nbytes = 8192;

myBuf = (char *)dtw_malloc(nbytes);

dtw_free((void *)myBuf);
```

dtw_getvar()

Usage

Retrieves the value of a configuration variable specified by *var_name* from the Net.Data initialization file. Net.Data owns the memory returned by dtw_getvar(); do not modify or free it.

Format

```
char *dtw_getvar(char *var_name)
```

Parameters

var_name	The name of the configuration variable to retrieve.

Examples

```
char *myBindFile;

myBindFile = dtw_getvar("BIND_FILE");
```

dtw_malloc()

Usage

Returns a pointer to storage that was allocated from Net.Data's run-time heap using dtw_malloc(). The storage is *nbytes* long. If Net.Data cannot return the requested storage, it returns a NULL pointer.

Format

```
void *dtw_malloc(long nbytes)
```

Parameters

nbytes	The number of bytes to allocate.

Examples

```
char *myBuf;
long  nbytes = 8192;

myBuf = (char *)dtw_malloc(nbytes);
```

dtw_row_SetCols()

Usage

Assigns the width of the row and allocates storage for the column headings. You can use the dtw_row_SetCols() utility function once for each row.

Format

```
int dtw_row_SetCols(void *row, int cols)
```

Parameters

row	A pointer to a newly created row which has not yet allocated any columns.
cols	The initial number of columns to allocate in the new row.

Examples

```
void *myRow;

rc = dtw_row_SetCols(myRow, 5);
```

dtw_row_SetV()

Usage

Assigns a table value. The caller of the dtw_row_SetV() utility function retains ownership of the memory pointed to by *src*. To delete the current table value, assign the value to NULL.

Format

```
int dtw_row_SetV(void *row, char *src, int col)
```

Parameters

row	A pointer to the row to modify.
src	A character string containing the new value to set.
col	The column number of the value to set.

Examples

```
void *myTable;
char *myFieldValue = "newValue";

rc = dtw_row_SetV(myRow, myFieldValue, 3);
```

dtw_strdup()

Usage

Allocates storage from Net.Data's run-time heap and copies the string specified by *string* into the allocated storage using dtw_malloc(). If Net.Data cannot return the requested storage, it returns a NULL pointer.

Format

```
char *dtw_strdup(char *string)
```

Parameters

string	A pointer to the string value to copy into the storage allocated.

Examples

```
char *myString = "This string will be duplicated.";
char *myDupString;

myDupString = dtw_strdup(myString);
```

dtw_table_AppendRow()

Usage

Adds one or more rows to the end of the table. Assign the table values of the new rows with the dtw_table_SetV() utility after rows are appended to the table.

Format

```
int dtw_table_AppendRow(void *table, int rows)
```

Parameters

table	A pointer to the table to be appended to.
rows	The number of rows to append.

Examples

```
void *myTable;

rc = dtw_table_AppendRow(myTable, 10);
```

dtw_table_Cols()

Usage

Returns the current number of columns in the table.

Format

```
int dtw_table_Cols(void *table)
```

Parameters

table — A pointer to the table whose current number of columns is returned.

Examples

```
void *myTable;
int currentColumns;

currentColumns = dtw_table_Cols(myTable);
```

dtw_table_Delete()

Usage

Deletes all of the column headings, table values, and the table object.

Format

```
int dtw_table_Delete(void *table)
```

Parameters

table	A pointer to the table to delete.

Examples

```
void *myTable;

rc = dtw_table_Delete(myTable);
```

dtw_table_DeleteCol()

Usage

Deletes one or more columns beginning at the column specified in *start_col*. To delete all of the rows and columns of a table, substitute the utility function dtw_table_Cols() for the *cols* parameter.

```
dtw_table_DeleteCol(table, 1, dtw_table_Cols());
```

Format

```
int dtw_table_DeleteCol(void *table, int start_col, int cols)
```

Parameters

table	A pointer to the table to modify.
start_col	The column number of the first column to delete.
rows	The number of columns to delete.

Examples

```
void *myTable;

rc = dtw_table_DeleteCol(myTable, 1, 10);
```

dtw_table_DeleteRow()

Usage

Deletes one or more rows beginning at the row specified in *start_row*.

Format

```
int dtw_table_DeleteRow(void *table, int start_row, int rows)
```

Parameters

table	A pointer to the table to modify.
start_row	The row number of the first row to delete.
rows	The number of rows to delete.

Examples

```
void *myTable;

rc = dtw_table_DeleteRow(myTable, 3, 10);
```

dtw_table_GetN()

Usage

Retrieves a column heading. Net.Data owns the memory pointed to by *dest*; do not modify or free it.

Format

```
int dtw_table_GetN(void *table, char **dest, int col)
```

Parameters

table	A pointer to the table from which a column heading is retrieved.
dest	A pointer to the character string to contain the column heading.
col	The column number of the column heading.

Examples

```
void *myTable;
char *myColumnHeading;

rc = dtw_table_GetN(myTable, &myColumnHeading, 5);
```

dtw_table_GetV()

Usage

Retrieves a value from a table. Net.Data owns the memory pointed to by *dest*; do not modify or free it.

Format

```
int dtw_table_GetV(void *table, char **dest, int row, int col)
```

Parameters

table	A pointer to the table from which a value is retrieved.
dest	A pointer to the character string that is to contain the value.
row	The row number of the value to retrieve.
col	The column number of the value to retrieve.

Examples

```
void *myTable;
char *myTableValue;

rc = dtw_table_GetV(myTable, &myTableValue, 3, 5);
```

dtw_table_InsertCol()

Usage

Inserts one or more columns after the specified column.

Format

```
int dtw_table_InsertCol(void *table, int after_col, int cols)
```

Parameters

table	A pointer to the table to modify.
after_col	The number of the column after which the new columns are to be inserted. To insert columns at the beginning of the table, specify 0.
cols	The number of columns to insert.

Examples

```
void *myTable;

rc = dtw_table_InsertCol(myTable, 3, 10);
```

dtw_table_InsertRow()

Usage

Inserts one or more rows after the specified row.

Format

```
int dtw_table_InsertRow(void *table, int after_row, int rows)
```

Parameters

table	A pointer to the table to modify.
after_row	The number of the row after which the new rows are inserted. To insert rows at the beginning of the table, specify 0.
rows	The number of rows to insert.

Examples

```
void *myTable;

rc = dtw_table_InsertRow(myTable, 3, 10);
```

dtw_table_MaxRows()

Usage

Returns the maximum number of rows allowed for the Net.Data table as defined by the dtw_table_New() utility function's parameter, *row_lim*.

Format

```
int dtw_table_MaxRows(void *table)
```

Parameters

table	A pointer to the table from which the maximum number of rows is returned.

Examples

```
void *myTable;
int maximumRows;

maximumRows = dtw_table_MaxRows(myTable);
```

dtw_table_New()

Usage

Creates a Net.Data table object and initializes all column headings and field values to NULL. The caller specifies the initial number of rows and columns, and the maximum number of rows. If the initial number of rows and columns is 0, you must use the dtw_table_SetCols() function to specify the number of fields in a row before any table function calls.

Format

```
int dtw_table_New(void **table, int rows, int cols, int row_lim)
```

Parameters

table	The name of the new table.
rows	The initial number of rows to allocate in the new table.
cols	The initial number of columns to allocate in the new table.
row_lim	The maximum number of rows this table can contain.

Examples

```
void *myTable;

rc = dtw_table_New(&myTable, 20, 5, 100);
```

dtw_table_QueryColnoNj()

Usage

Returns the column number associated with a column heading.

Format

```
int dtw_table_QueryColnoNj(void *table, char *name)
```

Parameters

table	A pointer to the table to query.
name	A character string specifying the column heading for which the column number is returned. If the column heading does not exist in the table, 0 is returned.

Examples

```
void *myTable;
int columnNumber;

columnNumber = dtw_table_QueryColnoNj(myTable, "column 1");
```

dtw_table_Rows()

Usage

Returns the current number of rows in the table.

Format

```
int dtw_table_Rows(void *table)
```

Parameters

table	A pointer to the table whose current number of rows is returned.

Examples

```
void *myTable;
     int currentRows;

     currentRows = dtw_table_Rows(myTable);
```

dtw_table_SetCols()

Usage

Sets the number of columns of the table and allocates storage for the column headings. Specify the column headings when the table is created; otherwise, you must specify them by calling this utility function before using any other table functions. You can only use the dtw_table_SetCols() utility function once for a table. Afterwards, use the dtw_table_DeleteCol() or dtw_table_InsertCol() utility functions.

Format

```
int dtw_table_SetCols(void *table, int cols)
```

Parameters

table	A pointer to a new table that has no columns or rows allocated.
cols	The initial number of columns to allocate in the new table.

Examples

```
void *myTable;

rc = dtw_table_SetCols(myTable, 5);
```

dtw_table_SetN()

Usage

Assigns a name to a column heading. The caller of the dtw_table_SetN() utility function retains ownership of the memory pointed to by the *src* parameter. To delete the column heading, assign the column heading value to NULL.

Format

```
int dtw_table_SetN(void *table, char *src, int col)
```

Parameters

table	A pointer to the table whose column heading is assigned.
src	A character string being assigned to the new column heading.
col	The number of the column.

Examples

```
void *myTable;
char *myColumnHeading = "newColumnHeading";

rc = dtw_table_SetN(myTable, myColumnHeading, 5);
```

dtw_table_SetV()

Usage

Assigns a value in a table. The caller of the dtw_table_SetV() utility function retains ownership of the memory pointed to by the *src* parameter. To delete the table value, assign the value to NULL.

Format

```
int dtw_table_SetV(void *table, char *src, int row, int col)
```

Parameters

table	A pointer to the table whose value is being assigned.
src	A character string assigned to the new value.
row	The row number of the new value.
col	The column number of the new value.

Examples

```
void *myTable;
char *myTableValue = "newValue";

rc = dtw_table_SetV(myTable, myTableValue, 3, 5);
```

Appendix A. Net.Data Technical Library

The Net.Data Technical Library is available from the Net.Data Web site at

`http://www.software.ibm.com/data/net.data/library.html`

Document	Description
• *Net.Data Administration and Programming Guide for OS/390* • *Net.Data Administration and Programming Guide for OS/2, Windows NT, and UNIX* • *Net.Data Administration and Programming Guide for OS/400*	Contains conceptual and task information about installing, configuring, and invoking Net.Data. Also describes how to write Net.Data macros, use Net.Data performance techniques, use Net.Data language environments, manage connections, and use Net.Data logging and traces for trouble shooting and performance tuning.
Net.Data Reference	Describes the Net.Data macro language, variables, and built-in functions.
Net.Data Language Environment Interface Reference	Describes the Net.Data language environment interface.
Net.Data Messages and Codes Reference	Lists Net.Data error messages and return codes.

Appendix B. Language Environment Template

Use this template to create your own language environments.

```
/*********************************************************************/
/*                                                                   */
/* File Name                                                         */
/*                                                                   */
/* Description                                                       */
/*                                                                   */
/* Functions                                                         */
/*                                                                   */
/* Entry Points                                                      */
/*                                                                   */
/* Change Activity                                                   */
/*                                                                   */
/* Flag    Reason      Date      Developer     Description           */
/* ------  ----------  --------  ------------  ---------------------- */
/*                                                                   */
/*********************************************************************/

/*-------------------------------------------------------------------*/
/* Includes                                                          */
/*-------------------------------------------------------------------*/
#include "dtwle.h"
```

Figure 2. Language Environment Template (Part 1 of 14)

```
#ifdef __MVS__
#pragma export(dtw_initialize)
#pragma export(dtw_execute)
#pragma export(dtw_getNextRow)
#pragma export(dtw_cleanup)
#endif

#ifdef _AIX_
//*------------------------------------------------------------------*/
/* Function                                                         */
/*    dtw_getFp                                                     */
/*                                                                  */
/* Purpose                                                          */
/*    Set function pointers to all Language Environment Interface   */
/*    routines being provided by this Language Environment. If a    */
/*    routine in the structure is not being provided, set that field */
/*    to NULL.                                                      */
/*                                                                  */
/* Format                                                           */
/*    int dtw_getFp(dtw_fp_t *func_pointer)                         */
/*                                                                  */
/* Parameters                                                       */
/*    func_pointer     A pointer to a structure which will contain  */
/*                     function pointers for all functions provided */
/*                     by this language environment.                */
/*                                                                  */
/* Returns                                                          */
/*    Success ....... 0                                             */
/*    Failure ....... -1                                            */
/*------------------------------------------------------------------*/
int dtw_getFp(dtw_fp_t *func_pointer)
{
    func_pointer->dtw_initialize_fp = dtw_initialize;
    func_pointer->dtw_execute_fp = dtw_execute;
    func_pointer->dtw_getNextRow_fp = dtw_getNextRow;
    func_pointer->dtw_cleanup_fp = dtw_cleanup;
    return 0;
}
#endif
```

Figure 2. Language Environment Template (Part 2 of 14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Function                                                          */
/*    dtw_initialize                                                 */
/*                                                                   */
/* Purpose                                                           */
/*                                                                   */
/* Format                                                            */
/*    int dtw_initialize(dtw_lei_t *le_interface)                    */
/*                                                                   */
/* Parameters                                                        */
/*    le_interface     A pointer to a structure containing the       */
/*                     following fields:                             */
/*                                                                   */
/*      function_name                                                */
/*      flags                                                        */
/*      exec_statement                                               */
/*      parm_data_array                                              */
/*      default_error_message                                        */
/*      le_opaque_data                                               */
/*      row                                                          */
/*                                                                   */
/* Returns                                                           */
/*    Success ....... 0                                              */
/*    Failure ....... 0                                              */
/*-------------------------------------------------------------------*/
int dtw_initialize(dtw_lei_t *le_interface)
{
    return rc;
}
```

Figure 2. Language Environment Template (Part 3 of 14)

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* Function                                                         */
/*    dtw_execute                                                   */
/*                                                                  */
/* Purpose                                                          */
/*                                                                  */
/* Format                                                           */
/*    int dtw_execute(dtw_lei_t *le_interface)                      */
/*                                                                  */
/* Parameters                                                       */
/*    le_interface     A pointer to a structure containing the      */
/*                     following fields:                            */
/*                                                                  */
/*      function_name                                               */
/*      flags                                                       */
/*      exec_statement                                              */
/*      parm_data_array                                             */
/*      default_error_message                                       */
/*      le_opaque_data                                              */
/*      row                                                         */
/*                                                                  */
/* Returns                                                          */
/*    Success ....... 0                                             */
/*    Failure ....... 0                                             */
/*------------------------------------------------------------------*/
int dtw_execute(dtw_lei_t *le_interface)
{
    /*--------------------------------------------------------------*/
    /* Determine if %exec statement was specified.                  */
    /*--------------------------------------------------------------*/
    if (le_interface->flags & DTW_STMT_EXEC) {
        /*----------------------------------------------------------*/
        /* Parse the %exec statement                                */
        /*----------------------------------------------------------*/
        rc = processExecStmt(le_interface->exec_statement);
        if (rc)
          {
          }
      }
    else {
        /*----------------------------------------------------------*/
        /* Parse the inline data                                    */
        /*----------------------------------------------------------*/
        rc = processInlineData(le_interface->exec_statement);
        if (rc)
          {
          }
      }
```

Figure 2. Language Environment Template (Part 4 of 14)

```
    /*--------------------------------------------------------------------*/
    /* Parse the input parameters                                         */
    /*--------------------------------------------------------------------*/
    rc = processInputParms(le_interface->parm_data_array);
    if (rc)
      {
      }
    /*--------------------------------------------------------------------*/
    /* Process the request                                                */
    /*--------------------------------------------------------------------*/
    rc = processRequest();
    if (rc)
      {
      }
    /*--------------------------------------------------------------------*/
    /* Process the output data                                            */
    /*--------------------------------------------------------------------*/
    rc = processOutputParms(le_interface->parm_data_array);
    if (rc)
      {
      }
    /*--------------------------------------------------------------------*/
    /* Process the return code and default error message                  */
    /*--------------------------------------------------------------------*/
    if (rc)
      {
        setErrorMessage(rc, &(le_interface->default_error_message));
      }
    /*--------------------------------------------------------------------*/
    /* Cleanup and exit program.                                          */
    /*--------------------------------------------------------------------*/
    return rc;
}
```

Figure 2. Language Environment Template (Part 5 of 14)

```
/*-----------------------------------------------------------------------*/
/*                                                                       */
/* Function                                                              */
/*    dtw_getNextRow                                                     */
/*                                                                       */
/* Purpose                                                               */
/*                                                                       */
/* Format                                                                */
/*    int dtw_getNextRow(dtw_lei_t *le_interface)                        */
/*                                                                       */
/* Parameters                                                            */
/*    le_interface     A pointer to a structure containing the          */
/*                     following fields:                                 */
/*                                                                       */
/*      function_name                                                    */
/*      flags                                                            */
/*      exec_statement                                                   */
/*      parm_data_array                                                  */
/*      default_error_message                                            */
/*      le_opaque_data                                                   */
/*      row                                                              */
/*                                                                       */
/* Returns                                                               */
/*    Success ....... 0                                                  */
/*    Failure ....... 0                                                  */
/*-----------------------------------------------------------------------*/
int dtw_getNextRow(dtw_lei_t *le_interface)
{
    return rc;
}
```

Figure 2. Language Environment Template (Part 6 of 14)

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* Function                                                         */
/*    dtw_cleanup                                                   */
/*                                                                  */
/* Purpose                                                          */
/*                                                                  */
/* Format                                                           */
/*    int dtw_cleanup(dtw_lei_t *le_interface)                      */
/*                                                                  */
/* Parameters                                                       */
/*    le_interface     A pointer to a structure containing the      */
/*                     following fields:                            */
/*                                                                  */
/*      function_name                                               */
/*      flags                                                       */
/*      exec_statement                                              */
/*      parm_data_array                                             */
/*      default_error_message                                       */
/*      le_opaque_data                                              */
/*      row                                                         */
/*                                                                  */
/* Returns                                                          */
/*    Success ....... 0                                             */
/*    Failure ....... 0                                             */
/*------------------------------------------------------------------*/
int dtw_cleanup(dtw_lei_t *le_interface)
{
    /*--------------------------------------------------------------*/
    /* Determine if this is normal or abnormal termination.         */
    /*--------------------------------------------------------------*/
    if (le_interface->flags & DTW_END_ABNORMAL) {
        /*----------------------------------------------------------*/
        /* Do abnormal termination cleanup.                         */
        /*----------------------------------------------------------*/
      }
    else {
        /*----------------------------------------------------------*/
        /* Do normal termination cleanup.                           */
        /*----------------------------------------------------------*/
      }

    return rc;
}
```

Figure 2. Language Environment Template (Part 7 of 14)

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* Function                                                         */
/*    processInputParms                                             */
/*                                                                  */
/* Purpose                                                          */
/*                                                                  */
/* Format                                                           */
/*    unsigned long processInputParms(dtw_parm_data_t *parm__data)   */
/*                                                                  */
/* Parameters                                                       */
/*    dtw_parm_data_t *parm_data                                    */
/*                                                                  */
/* Returns                                                          */
/*    Success ..... 0                                               */
/*    Failure .....                                                 */
/*                                                                  */
/*------------------------------------------------------------------*/
unsigned long processInputParms(dtw_parm_data_t *parm_data)
{
    /*--------------------------------------------------------------*/
    /* Loop through all the variables in the parameter data array.    */
    /* The array is terminated by a NULL entry, meaning the parm_name */
    /* field is set to NULL, the parm_value field is set to NULL, and */
    /* the parm_descriptor field is set to 0. However, the only valid */
    /* check for the end of the parameter data array is to check      */
    /* parm_descriptor == 0, since the parm_name field is NULL when a */
    /* literal string is passed in, and the parm_value field is set   */
    /* to NULL when an undeclared variable is passed in.              */
    /*--------------------------------------------------------------*/
    for (; parm_data->parm_descriptor != 0; ++parm_data) {
```

Figure 2. Language Environment Template (Part 8 of 14)

```
        /*-------------------------------------------------------------*/
        /* Determine the usage of each input parameter.                */
        /*-------------------------------------------------------------*/
        switch(parm_data->parm_descriptor & DTW_USAGE) {

            case(DTW_IN):
                /*-----------------------------------------------------*/
                /* Determine the type of each input parameter.         */
                /*-----------------------------------------------------*/
                switch (parm_data->parm_descriptor & DTW_TYPE) {
                    case DTW_STRING:
                        break;
                    case DTW_TABLE:
                        break;
                    default:
                        /*---------------------------------------------*/
                        /* Internal error - unknown data type          */
                        /*---------------------------------------------*/
                        break;
                  }
                break;

            case(DTW_OUT):
                break;

            case(DTW_INOUT):
                break;

            default:
                /*-----------------------------------------------------*/
                /* Internal error - unknown usage                      */
                /*-----------------------------------------------------*/
                break;
          }
      }
    return rc;
}
```

Figure 2. Language Environment Template (Part 9 of 14)

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* Function                                                         */
/*    processOutputParms()                                          */
/*                                                                  */
/* Purpose                                                          */
/*                                                                  */
/* Format                                                           */
/*    unsigned long processOutputParms(dtw_parm_data_t *parm_data)    */
/*                                                                  */
/* Parameters                                                       */
/*    dtw_parm_data_t *parm_data                                    */
/*                                                                  */
/* Returns                                                          */
/*    Success ........ 0                                            */
/*    Failure ........ -1                                           */
/*                                                                  */
/*------------------------------------------------------------------*/
unsigned long processOutputParms(dtw_parm_data_t *parm_data) {
    /*--------------------------------------------------------------*/
    /* Get output data in some language environment-specific manner.  */
    /* This is entirely dependent on what the language environment    */
    /* is interfacing to, and how the LE chooses to interface to it.  */
    /*--------------------------------------------------------------*/
```

Figure 2. Language Environment Template (Part 10 of 14)

```
/    /*-------------------------------------------------------------*/
    /* Loop through all the parms in the parameter data array,       */
    /* looking for output parameters.                                */
    /*---------------------------------------------------------------*/
    for (; parm_data->parm_descriptor != 0; ++parm_data) {

        /*-----------------------------------------------------------*/
        /* Determine usage of each parameter.                        */
        /*-----------------------------------------------------------*/
        if (pd_i->parm_descriptor & DTW_OUT) {
            /*-------------------------------------------------------*/
            /* Determine the type of each input parameter.           */
            /*-------------------------------------------------------*/
            switch (pd_i->parm_descriptor & DTW_TYPE) {
                case DTW_STRING:
                    /*-----------------------------------------------*/
                    /* Give a string parameter a new value. If the   */
                    /* parameter value is not currently NULL, the    */
                    /* storage must be freed using an LE interface   */
                    /* utility function if it was allocated by       */
                    /* Net.Data.                                     */
                    /*-----------------------------------------------*/
                    if (parm_data->parm_value != NULL)
                        dtw_free(parm_data->parm_value);
                    parm_data->parm_value = dtw_strdup(newValue);
                    break;
                case DTW_TABLE:
                    /*-----------------------------------------------*/
                    /* Change the size of a table parameter. Use the */
                    /* LE interface utility functions to modify the  */
                    /* table object.                                 */
                    /*-----------------------------------------------*/
                    /*-----------------------------------------------*/
                    /* First get the pointer to the table object.    */
                    /*-----------------------------------------------*/
                    void *myTable = (void *) parm_data->parm_value;
```

Figure 2. Language Environment Template (Part 11 of 14)

```
/*----------------------------------------------*/
/* Next get the current size of the table.      */
/*----------------------------------------------*/
cols = dtw_table_Cols(myTable);
rows = dtw_table_Rows(myTable);
/*----------------------------------------------*/
/* Now set the new size (assumes the new size   */
/* values are valid).                           */
/*----------------------------------------------*/

/*----------------------------------------------*/
/* Set the columns first.                       */
/*----------------------------------------------*/
if (cols > newColValue)
  {
    dtw_table_DeleteCol(myTable,
                        newColValue + 1,
                        cols - newColValue);
  }
else if (cols < new_col_value)
  {
    dtw_table_InsertCol(myTable,
                        cols,
                        newColValue - cols);
  }

/*----------------------------------------------*/
/* Now set the rows.                            */
/*----------------------------------------------*/
if (newColValue > 0) {
    if (rows > newRowValue)
      {
        dtw_table_DeleteRow(myTable,
                            newRowValue + 1,
                            rows - newRowValue);
      }
    else if (rows < new_row_value)
      {
        dtw_table_InsertRow(myTable,
                            rows,
                            newRowValue - rows);
      }
  }
```

Figure 2. Language Environment Template (Part 12 of 14)

```
                    /*-----------------------------------------------*/
                    /* Now get the last row/column value.            */
                    /*-----------------------------------------------*/
                    dtw_table_GetV(myTable,
                                   &myValue;,
                                   newRowValue,
                                   newColValue);

                    /*-----------------------------------------------*/
                    /* Delete the last row/column value.             */
                    /*-----------------------------------------------*/
                    dtw_table_SetV(myTable,
                                   NULL,
                                   newRowValue,
                                   newColValue);

                    /*-----------------------------------------------*/
                    /* Set the last row/column value.                */
                    /*-----------------------------------------------*/
                    dtw_table_SetV(myTable,
                                   dtw_strdup(myNewValue),
                                   newRowValue,
                                   newColValue);

                    break;
                default:
                    /*-----------------------------------------------*/
                    /* Internal error - unknown data type            */
                    /*-----------------------------------------------*/
                    break;
            }
        }
    }

  return 0;
}
```

Figure 2. Language Environment Template (Part 13 of 14)

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* Function                                                         */
/*    setErrorMessage()                                             */
/*                                                                  */
/* Purpose                                                          */
/*                                                                  */
/* Format                                                           */
/*    unsigned long setErrorMessage(int returnCode,                 */
/*                                  char **defaultErrorMessage)     */
/*                                                                  */
/* Parameters                                                       */
/*    int    returnCode                                             */
/*    char **defaultErrorMessage                                    */
/*                                                                  */
/* Returns                                                          */
/*    Success ........ 0                                            */
/*    Failure ........ -1                                           */
/*                                                                  */
/*------------------------------------------------------------------*/
unsigned long setErrorMessage(int returnCode,
                              char **defaultErrorMessage)
{

    /*--------------------------------------------------------------*/
    /* Set the default error message based on the return code.      */
    /*--------------------------------------------------------------*/
    switch(returnCode) {
        case LE_SUCCESS:
            break;
        case LE_RC1:
            *defaultErrorMessage = dtw_strdup(LE_RC1_MESSAGE_TEXT);
            break;
        case LE_RC2:
            *defaultErrorMessage = dtw_strdup(LE_RC2_MESSAGE_TEXT);
            break;
        case LE_RC3:
            *defaultErrorMessage = dtw_strdup(LE_RC3_MESSAGE_TEXT);
            break;
        case LE_RC4:
            *defaultErrorMessage = dtw_strdup(LE_RC4_MESSAGE_TEXT);
            rc = LE_RC1INTERNAL;
            break;
      }
    return 0;
}
```

Figure 2. Language Environment Template (Part 14 of 14)

Appendix C. Build File Examples

The following examples are provided as reference for building an LE on various operating systems. Because your particular operating system and environment has its own unique characteristics, these examples should only be used for reference. When you build your shared libraries, please consult the documentation specific to the compiler you are using on your operating system.

Sample OS/390 JCL

```
//BLDUSER  JOB ,'BLDUSER ',TIME=1,
// MSGCLASS=H,CLASS=A,
// USER=IBMUSER,MSGLEVEL=(1,1)
//*********************************************************************
//*  COMPILE STEP: C++ DLL     (USER-WRITTEN LE)                      *
//*                C/C++ FOR MVS/ESA(R) COMPILER V3 R2.0              *
//*                AD/CYCLE LE/370 V1 R7.0                            *
//*********************************************************************
//COMPILE EXEC PGM=CBC320PP,REGION=32M,
//          PARM=('/CXX SO,OPT,EXP,SE(''CEEV1R70.SCEEH.+'')',
//          'DEF(_XOPEN_SOURCE_EXTENDED)')
//STEPLIB   DD DSN=CEEV1R70.SCEERUN,DISP=SHR
//          DD DSN=CBCV3R20.SCBC3CMP,DISP=SHR
//SYSMSGS   DD DUMMY,DSN=CBCV3R20.SCBC3MSG(EDCMSGE),DISP=SHR
//SYSXMSGS  DD DUMMY,DSN=CBCV3R20.SCBC3MSG(CBCMSGE),DISP=SHR
//SYSIN     DD DSN=IBMUSER.NETDATA.USERLANG.C(USERLANG),DISP=SHR
//SYSLIB    DD DSN=CBCV3R20.SCLB3H.H,DISP=SHR
//USERLIB   DD DSN=IBMUSER.NETDATA.H,DISP=SHR
//SYSLIN    DD DSN=IBMUSER.NETDATA.USERLANG.OBJ(USERLANG),DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSCPRT   DD SYSOUT=*
//SYSTERM   DD DUMMY
//SYSUT1    DD DSN=&&SYSUT1;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT2    DD DSN=&&SYSUT2;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT3    DD DSN=&&SYSUT3;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4    DD DSN=&&SYSUT4;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5    DD DSN=&&SYSUT5;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT6    DD DSN=&&SYSUT6;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7    DD DSN=&&SYSUT7;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8    DD DSN=&&SYSUT8;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9    DD DSN=&&SYSUT9;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
```

```
//SYSUT10   DD SYSOUT=*
//SYSUT14   DD DSN=&&SYSUT14;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT15  DD  SYSOUT=*
//*
//*********************************************************************
//*  PRELINK STEP: C++ DLL     (USER-WRITTEN LE)                      *
//*              C/C++ FOR MVS/ESA COMPILER V3 R1.0                   *
//*              AD/CYCLE LE/370 V1 R7.0                              *
//*********************************************************************
//PLKED EXEC PGM=EDCPRLK,REGION=32M,COND=(0,NE,COMPILE),
//          PARM='MAP,UPCASE,MEMORY,DLLNAME(userdll)'
//STEPLIB   DD DSN=CEEV1R70.SCEERUN,DISP=SHR
//SYSMSGS   DD DSN=CEEV1R70.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB    DD DSN=CEEV1R70.SCEECPP,DISP=SHR
//SYSMOD    DD DSN=IBMUSER.NETDATA.USERLANG.DLLP(PRLKUSER),
//             DISP=SHR
//OBJECT    DD DSN=IBMUSER.NETDATA.USERLANG.OBJ,DISP=SHR
//SYSOUT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSDEFSD  DD DSN=IBMUSER.NETDATA.USERLANG.DEFSD(USEREXP),
//             DISP=SHR
//SYSIN     DD DSN=IBMUSER.NETDATA.DEFSD(DTWLESHR),DISP=SHR
//          DD DSN=CBCV3R20.SCLB3SID(COMPLEX),DISP=SHR
//          DD DSN=CBCV3R20.SCLB3SID(APPSUPP),DISP=SHR
//          DD DSN=CBCV3R20.SCLB3SID(COLLECT),DISP=SHR
//          DD *
    INCLUDE OBJECT(USERLANG)
//*
//*********************************************************************
//*  LINK STEP: C++ DLL     (FFI LANGUAGE ENVIRONMENT)                *
//*           C/C++ FOR MVS/ESA COMPILER V3 R2.0                      *
//*           AD/CYCLE LE/370 V1 R7.0                                 *
//*********************************************************************
//LKED EXEC PGM=HEWL,REGION=2048K,COND=(4,LT,PLKED),
//          PARM='MAP,LIST,XREF,RENT,REUS,COMPAT=PM2'
//SYSLIB    DD DSN=CEEV1R70.SCEELKED,DISP=SHR
//SYSLIN    DD DSN=IBMUSER.NETDATA.USERLANG.DLLP(PRLKUSER),
//             DISP=SHR
//          DD *
            NAME USERDLL(R)
//SYSLMOD   DD DSN=IBMUSER.NETDATA.USERLANG.DLL,DISP=SHR
//SYSUT1    DD DSN=&&SYSUT1;,UNIT=SYSALLDA,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT  DD SYSOUT=*
//*
//*********************************************************************
//*  COPY STEP:  C++ DLL     (FFI LANGUAGE ENVIRONMENT)               *
//*            C/C++ FOR MVS/ESA COMPILER V3 R2.0                     *
//*            AD/CYCLE LE/370 V1 R7.0                                *
//*********************************************************************
//COPY EXEC PGM=IEWBLINK,REGION=500K,COND=(0,NE,LKED),
//          PARM='LIST,REUS,RENT,NCAL,LET,MAP,CASE=MIXED,COMPAT=PM2'
//SYSPRINT  DD SYSOUT=*
//INLIB     DD DSN=IBMUSER.NETDATA.USERLANG.DLL,DISP=SHR
//*
//SYSLMOD   DD PATH='/usr/lpp/netdata/cgi-bin',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=(SIRWXO,SIRWXG,SIRWXU)
```

```
//*
//SYSLIN    DD *
            INCLUDE INLIB(USERDLL)
            ENTRY CEESTART
            NAME userdll(R)
//*
```

Sample makefile (OS/390-specific)

```
### Target
TARGET          = userlang

### C Compiler
CC              = c89
CFLAGS          = -D_XOPEN_SOURCE_EXTENDED=1 -O -I/u/USER01/netdata/include

### C++ Compiler
CCC             = c++
CCFLAGS         = -D_XOPEN_SOURCE_EXTENDED=1 -O -I/u/USER01/netdata/include

### Linker/Loader
LD              = c++
LDFLAGS         = $(CCFLAGS) -W l,dll

### Sources Headers and Objects
OBJS            = userlang.o
HDRS            = /u/USER01/netdata/include/dtwle.h userlang.h


#######################
### Program Libraries #
#######################

LIBS = /u/USER01/netdata/defsd/dtwleshr.x

########################
### Additional Targets #
########################

all:        $(TARGET)

$(TARGET):  $(OBJS) $(LIBS)
            echo "Linking $(TARGET) ..."
            $(LD) $(LDFLAGS) -o $(TARGET) $(OBJS) $(LIBS)
            echo "done"

userlang.o: $(HDRS) userlang.c
            $(CCC) $(CCFLAGS) -c userlang.c

clean:
            rm -f $(OBJS) $(TARGET)
```

Sample OS/400 CL

Assuming the following conditions, use the subsequent steps to build a Language Environment on AS/400:

- SRC is the source file (written in C).
- MYLE contains the exportable procedure, `dtw_execute`.
- The file, QSRVSRC, member MYLEEXP, contains the specifications for exporting the procedure `dtw_execute`.

1. Create the module:

   ```
   CRTCMOD MODULE(MYLIB/MYLE) SRCFILE(MYLIB/SRC)
   ```

2. Create the service program:

```
CRTSRVPGM SRVPGM(MYLIB/MYLE) MODULE(MYLIB/MYLE)
                             SRCFILE(MYLIB/QSRVSRC) SRCMBR(MYLEEXP)
                             BNDSRVPGM(QTCP/QTMHLE)
```

QTCP/QTMHLE is the service program that contains all the Net.Data bindable APIs. On V4R3 and subsequent releases, although QTCP/QTMHLE is available for use, you should use QHTTPSVR/QTMJLE.

Appendix D. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
AS/400
DB2
DB2 Universal Database
DRDA
DataJoiner
IBM
IMS
Language Environment
MVS/ESA
Net.Data
OS/2
OS/390
OS/400
OpenEdition

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

absolute path. The full path name of an object. Absolute path names begins at the highest level, or ″root″ directory (which is identified by the forward slash (/) or back slash (\) character).

API. Application programming interface. Net.Data supports three Web server APIs for improved performance over CGI processes.

applet. A Java program included in an HTML page. Applets work with Java-enabled browsers, such as Netscape Navigator, and are loaded when the HTML page is processed.

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.

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.

commitment control. The establishment of a boundary within the process that Net.Data is running under where operations on resources are part of a unit of work.

Common Gateway Interface (CGI). A standardized way for a Web server to pass control to an application program and receive data back.

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*

current working directory. The default directory of a process from which all relative path names are resolved.

database. A collection of tables, or a collection of table spaces and index spaces.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

DATALINK. A DB2 data type that enables logical references from the database to a file stored outside the database.

data type. An attribute of columns and literals.

DBCLOB. Double-byte character large object.

DBMS. Database management system.

Domino *Go* Web server. The Web server offered by Lotus Corp. and IBM, that offers both regular and secure connections. ICAPI and GWAPI are the interfaces provided with this server.

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.

GWAPI. *Go* Web server API.

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. *See* .

Internet. An international public TCP/IP computer network.

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.

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.

path name. Tells the system how to locate an object. The path name is expressed as a sequence of directory names followed by the name of the object. Individual directories and the object name are separated by a forward slash (/) or back slash (\) character.

Perl. An interpreted programming language.

persistence. The state of keeping an assigned value for an entire transaction, where a transaction spans multiple Net.Data invocations. Only variables can be persistent. In addition, operations on resources affected by commitment control are kept active until an explicit commit or rollback is done, or when the transaction completes.

port. A 16-bit number used to communicate between TCP/IP and a higher level protocol or application.

registry. A repository where strings can be stored and retrieved.

relative path name. A path name that does not begin at the highest level, or ″root″ directory. The system assumes that the path name begins at the process's current working directory.

TCP/IP. Transmission Control Protocol / Internet Protocol.

transaction. One Net.Data invocation. If persistent Net.Data is used, then a transaction can span multiple Net.Data invocations.

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`.

unit of work. A recoverable sequence of operations that are treated as one atomic operation. All operations within the unit of work can be completed (commited) or undone (rolled back) as if the operations are a single operation. Only operations on resources that are affected by commitment control can be committed or rolled back.

Web server. A computer running HTTP server software, such as Internet Connection.

Index

R

S

T

U

V

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.