Net.Data

言語環境解説書

SB88-7410-00

Net.Data

言語環境解説書

SB88-7410-00

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、 59ページの『付録D. 特記事項』に記載する一般情報をお読みください。

原　典： Net.Data
Language Environment Interface
Reference

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 1999.5

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

© Copyright International Business Machines Corporation 1997, 1998. All rights reserved.

Translation: © Copyright IBM Japan 1999

目次

まえがき v
Net.Data について v
本書について vi
本書の対象読者 vi
本書の例について vi

Net.Data の言語環境について vii

第1章 新しい言語環境の作成 1
共用ライブラリーの作成 1
指定すべき言語環境インターフェース 2
入力パラメーターの処理 2
ユーザー要求の処理. 3
出力パラメーターの処理 3
エラー条件の伝達 3
言語環境通信構造体. 4
dtw_lei_t 構造体 4
dtw_parm_data_t 構造体 6
言語環境インターフェース関数 7
dtw_initialize() 8
dtw_execute() 8
dtw_getNextRow(). 9
dtw_cleanup() 10
言語環境ステートメントの設計 10
ENVIRONMENT ステートメントの構文 11
ENVIRONMENT ステートメントの例 12

第2章 言語環境のプログラミング・インターフェース・ユーティリティー関数 . 13
言語環境ユーティリティー関数 13
メモリー管理用のユーティリティー関数 13
構成変数を管理するためのユーティリティー関数 14
表操作用のユーティリティー関数. 14
行操作用のユーティリティー関数. 15
ユーティリティー関数の構文参照. 15
dtw_free(). 16
dtw_getvar() 17
dtw_malloc(). 18
dtw_row_SetCols(). 19
dtw_row_SetV() 20
dtw_strdup() 21
dtw_table_AppendRow(). 22
dtw_table_Cols() 23
dtw_table_Delete(). 24
dtw_table_DeleteCol() 25
dtw_table_DeleteRow() 26
dtw_table_GetN() 27
dtw_table_GetV() 28
dtw_table_InsertCol() 29
dtw_table_InsertRow() 30

© Copyright IBM Corp. 1997, 1998

dtw_table_MaxRows() 31
dtw_table_New() 32
dtw_table_QueryColnoNj() 33
dtw_table_Rows() 34
dtw_table_SetCols() 35
dtw_table_SetN() 36
dtw_table_SetV() 37

付録A. Net.Data テクニカル・ライブラリー 39

付録B. 言語環境のテンプレート 41

付録C. ファイル例の作成 55
OS/390 JCL の例. 55
makefile の例 (OS/390 固有) 57
OS/400 CL の例 58

付録D. 特記事項 59
商標 60

用語集. 61

索引 63

まえがき

Net.Data® をお買上げいただきありがとうございます。本製品は、動的な Web ページを作成するための IBM® 開発ツールです。 Net.Data を使用すると、さまざまなデータ・ソースからデータを取り込んだり、既知のプログラム言語が持つ長所を生かしたりして、動的コンテンツを含む Web ページを迅速に開発することができます。

Net.Data について

IBM の Net.Data プロダクトを用いると、DB2、IMS、ODBC 対応のデータベース、および DRDA を介してアクセスできるデータベースを含め、リレーショナル・データベース管理システムおよび非リレーショナル・データベース管理システム (DBMS) のデータを使用して、また Java、JavaScript、Perl、C、C++、REXX などプログラミング言語で作成されたアプリケーションを使用して、動的な Web ページを作成することができます。 Net.Data ファミリー・プロダクトは、 Windows NT、AIX、OS/2、OS/390、OS/400、HP-UX、Sun Solaris、Santa Cruz Operating System (SCO)、および Linux オペレーティング・システム環境のマシンで同じ機能を提供します。

Net.Data は、Web サーバー上のミドルウェアとして実行するマクロ処理プログラムです。 マクロ と呼ばれる Net.Data アプリケーション・プログラムを作成することができます。 Net.Data は、このマクロを解釈して、ユーザー、データベースの現状、その他のデータ・ソース、既存のビジネス論理、およびマクロ中に組み込むその他の係数からの入力データに基づいて、カスタマイズされた内容の動的な Web ページを作成します。

要求は URL (uniform resource locator) の形式で Netscape Navigator や Internet Explorer などのブラウザーから Web サーバーに流れ、Web サーバーはその要求を Net.Data に送って実行します。 Net.Data はこのマクロを探し出して実行し、Web ページを作成して、ユーザーが作成した関数に基づいてその Web ページをカスタマイズします。これらの関数は以下のことを行うことができます。

- C、C++、RPG、COBOL、JAVA、Perl、または REXX プログラミング言語で作成された (しかし、これらの言語に限定される) アプリケーションにビジネス論理をカプセル化する
- DB2 などのデータベースにアクセスする
- フラット・ファイルなどのその他のデータ・ソースにアクセスする

Net.Data は、この Web ページを Web サーバーに渡し、 Web サーバーはネットワークを介してそのページをブラウザーに渡し、そのページはブラウザーで表示されます。

Net.Data は、HTTP (HyperText Transfer Protocol) やコモン・ゲートウェイ・インターフェース (CGI) などのインターフェースを使用するように設定された、サーバー環境で使用することができます。 HTTP は、ブラウザーと Web サーバー間の対話用の業界標準インターフェースで、 CGI は、Net.Data などのゲートウェイ・アプリケーションを Web サーバーが起動できるようにするための業界標準インターフェースです。

© Copyright IBM Corp. 1997, 1998

これらのインターフェースにより、好みのブラウザーもしくは Web サーバーを選んで Net.Data と一緒に使用することができます。

パフォーマンスを向上させるため、Net.Data は各種の Web サーバーのアプリケーション・プログラミング・インターフェース (API) をサポートしています。さらに、Net.Data は Java サーブレットとして起動することもできます。

本書について

本書では、Net.Data に合った独自のカスタム言語環境を開発する場合に使用できる、Net.Data 言語環境インターフェース (LEI) について説明します。

本書では、発表されていても、まだ利用可能でない製品または機能について言及する場合があります。

Net.Data のサンプル・マクロ、デモ、および本書の最新版などについての詳細は、以下の WWW サイトで見ることができます。

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

本書の対象読者

個々の会社のニーズに合うように Net.Data の機能を拡張したい方は、本書を読むことで、Net.Data に合った独自の言語環境を作成することができます。

本書で説明する概念を理解するには、次の情報をよく知っていることが必要です。

- C プログラミング言語
- *Net.Data 管理およびプログラミングの手引き* および *Net.Data 解説書* の内容

本書の例について

本書で使用されている例は、特定の概念を説明するために単純化されたものであり、考えられるすべてのケースを考慮したものではありません。例の中には、単独では動作しない断片的なものもあります。

Net.Data の言語環境について

Net.Data は、後から付け加える形で新しいプログラム言語やデータベース・インターフェースを追加できるように設計されています。これらのインターフェースは言語環境と呼ばれ、DLL または共用ライブラリーとしてアクセスされます。言語環境からは、動的 Web ページをサポートするアプリケーションおよびデータベースにアクセスすることができます。関数呼び出しを使用して言語環境を呼び出すことで、これらの言語環境が提供する、ユーザーのビジネス・アプリケーションに役立つ機能を使用することができます。 たとえば、ODBC データベースに直接アクセスして、 Perl 言語環境を使用して Perl スクリプトを実行したり、 Java Applet 言語環境を呼び出して Java アプレットを実行したりできます。

Net.Data 初期設定ファイルは、各言語環境名と DLL または共用ライブラリーを関連付けます。それぞれの言語環境は、 Net.Data によって定義されたインターフェースの標準セットをサポートしなければなりません。 Net.Data は、その言語環境を指定している FUNCTION ブロックに対する関数呼び出しが最初に検出されたときに、初期設定ファイルで指定されている DLL または共用ライブラリーをロードします。

Net.Data は、Net.Data マクロの構文解析、Net.Data 変数の保守、言語環境との通信、REPORT および MESSAGE ブロックの仕様に従った出力のフォーマットを行います。言語環境は、Net.Data に定義されたインターフェースのサポート、 Net.Data パラメーターへの言語処理プログラムによる言語別の様式でのアクセス、言語インタープリターの呼び出し、言語インタープリターからの言語別の様式による変数の受け取りを行います。

viiiページの図1 は、Net.Data と言語環境との相互作用を示しています。

© Copyright IBM Corp. 1997, 1998

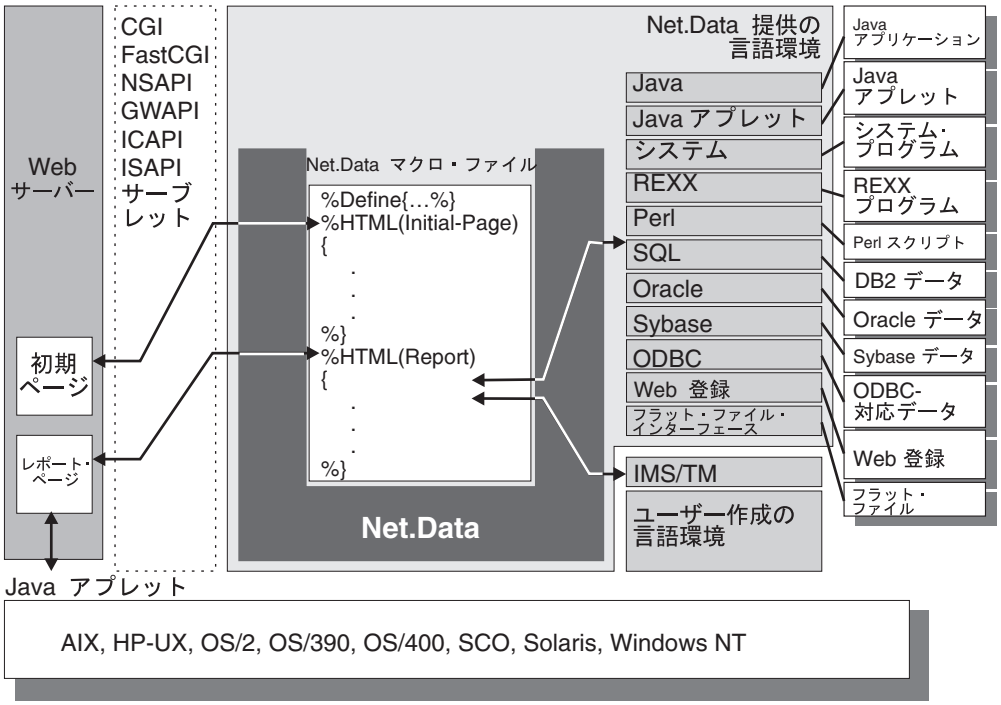

図 1. Net.Data と言語環境

本書では、新しい言語環境を作成する場合に使用する、 Net.Data 言語環境インターフェースについて説明します。 Net.Data 提供の言語環境については、オペレーティング・システムの *Net.Data 管理およびプログラミングの手引き* の言語環境の章を参照してください。

第1章 新しい言語環境の作成

Net.Data は、言語環境プログラムを、DLL ファイル、共用ライブラリー、またはサービス・プログラムとしてアクセスする (いずれであるかは、オペレーティング・システム環境によって異なります)、交換可能なプログラム言語およびデータベース・インターフェースとして使用します。本書では、*共用ライブラリー* という用語は、一般にこれらのタイプのファイルを指すときに使用します。 Net.Data には、一連の言語環境が用意されていますが、それらの言語環境がアプリケーションのニーズに合わない場合は、 Net.Data 言語環境インターフェースを使用して、独自の言語環境を作成することができます。

新しい言語環境を作成する場合は、次のステップを行ってください。

- 言語環境に指定しなければならないインターフェースと関数を決定します。 dtw_execute() インターフェースを指定し、かつ指定したすべてのインターフェースが dtwle.h C 言語見出しで定義されたプロトタイプと正確に一致していなければなりません。
- 提供したい一連の言語環境インターフェース・ルーチンをインプリメントする共用ライブラリーを作成します。共用ライブラリーの作成方法については、コンパイラーのマニュアルを参照してください。
- Net.Data が呼び出すことができるように、すべてのインターフェースを共用ライブラリーから外部で使用できるようにします。
- ENVIRONMENT 構成ステートメントを決定して、 Net.Data 初期設定ファイルに追加します。
- 新しい言語環境を使用する Net.Data マクロに関数を追加します。

新しい言語環境を作成することを決める前に、 Net.Data 提供の言語環境が必要条件を満たしているかどうかを判断してください。

本章では、言語環境の設計方法を説明します。

- 『共用ライブラリーの作成』
- 4ページの『言語環境通信構造体』
- 7ページの『言語環境インターフェース関数』
- 10ページの『言語環境ステートメントの設計』

言語環境プログラミング・インターフェースの詳細については、 13ページの『第2章 言語環境のプログラミング・インターフェース・ユーティリティー関数』 を参照してください。

共用ライブラリーの作成

言語環境を作成するときは、 41ページの『付録B. 言語環境のテンプレート』 に示されているテンプレートから、 Net.Data が言語環境との間でパラメーターをやり取りする際に使用する環境インターフェース関数や通信構造を利用することができます。

© Copyright IBM Corp. 1997, 1998

以下の節では、関数と構造体に関する概念と設計上の問題について説明します。言語環境インターフェースで提供されるユーティリティーについては、13ページの『第2章 言語環境のプログラミング・インターフェース・ユーティリティー関数』 で説明します。

- 『指定すべき言語環境インターフェース』
- 『入力パラメーターの処理』
- 3ページの『ユーザー要求の処理』
- 3ページの『出力パラメーターの処理』
- 3ページの『エラー条件の伝達』

指定すべき言語環境インターフェース

言語環境を作成するときは、指定するインターフェースを決めなければなりません。言語環境に行わせたい内容によって、何を選択するかが決まります。たとえば、言語環境がデータベース・データにアクセスする場合には、スクリプト言語の場合とは異なる選択になります。以下の節では、Net.Data 言語環境インターフェースについて説明します。

dtw_execute()

dtw_execute() インターフェースは、マクロから入力パラメーターを渡すために指定しなければなりません。これは、言語環境ごとに必要となる唯一のインターフェースです。 Net.Data は、すべての入力パラメーターを、言語環境通信構造体 dtw_lei_t を介して dtw_execute() に渡します。

dtw_initialize()

dtw_initialize() インターフェースは、データを割り当てる場合または初期化する場合に指定します。 Net.Data がこのインターフェースを呼び出すのは各マクロ起動につき 1 回だけで、言語環境への最初の関数呼び出しの前に行われます。言語環境への関数呼び出しがなければ、Net.Data は dtw_initialize() インターフェースを呼び出しません。

dtw_cleanup()

dtw_cleanup() インターフェースは、dtw_initialize() インターフェースを指定して、資源を解放したい場合に指定します。

dtw_getNextRow()

dtw_getNextRow() インターフェースは、データベース言語環境、もしくは一度に 1 行ずつデータを処理できる言語環境の一部として指定します。このインターフェースが呼び出されるのは、Net.Data が OS/400® または OS/390 オペレーティング・システム上で実行されている場合です。

入力パラメーターの処理

Net.Data 言語環境は、dtw_execute() インターフェースを使用してパラメーターを受け取り、処理します。 dtw_execute() インターフェースは、 Net.Data が言語環境と通信する際に使用する dtw_lei_t 構造体を処理します。言語環境を作成するときは、入力パラメーターの処理に関する以下の推奨事項を使用してください。

- Net.Data 初期設定ファイルで、言語環境に対する ENVIRONMENT ステートメントに暗黙のパラメーターを指定します。 Net.Data は、実行される FUNCTION ブロックのマクロ書き込みプロセスによって指定されたパラメーターを渡した後で、このすべての関数呼び出しで指定されたパラメーターを言語環境に渡します。
- dtw_execute() インターフェースへの入力パラメーターを、dtw_lei_t 構造体の一部として受け取ります。マクロ書き込みプロセスは、Net.Data マクロの FUNCTION ブロック定義でパラメーターを指定する際に、それを Net.Data が渡す順序を決定します。

41ページの『付録B. 言語環境のテンプレート』 のプログラム・テンプレートの processInputParms() ルーチンで、入力パラメーターを処理する方法の 1 つを示します。

ユーザー要求の処理

言語環境がユーザー要求を処理する方法は、言語環境が要求を受け取る方法によって異なります。 Net.Data には、要求を言語環境に伝達するためのいくつかの異なる方法があります。

- FUNCTION ブロックで指定された関数名を使用。関数呼び出しのたびに、Net.Data は、dtw_lei_t 構造体の function_name フィールドで関数名を言語環境に渡します。
- FUNCTION ブロックのパラメーター・リストを使用。パラメーター・リスト内のパラメーターによってユーザー要求を示すことができることを指定できます。関数呼び出しのたびに、Net.Data は、dtw_lei_t 構造体の parm_data_array フィールドでパラメーターを言語環境に渡します。
- FUNCTION ブロックの executable-statements セクションを使用。関数呼び出しのたびに、Net.Data は、dtw_lei_t 構造体の exec_statement フィールドで、 FUNCTION ブロックに指定された任意の実行可能ステートメントを言語環境に渡します。

出力パラメーターの処理

出力パラメーターの処理に使用する方法は、言語環境とそれがユーザー要求を処理する方法によってまったく異なります。しかし、言語環境に Net.Data マクロに戻さなければならないデータがある場合には、 dtw_lei_t 構造体の parm_data_array フィールドで渡されたパラメーターの値を変更するように言語環境を設計することができます。 41ページの『付録B. 言語環境のテンプレート』 のプログラム・テンプレートの processOutputParms() ルーチンで、出力パラメーターを処理する 1 つの考えられる方法と、ストリングと表の両方のパラメーター値を設定する方法を示します。

エラー条件の伝達

関数呼び出しの成否は、暗黙 Net.Data マクロ変数 RETURN_CODE を介して伝達することができます。この変数は、dtw_execute() インターフェースへの呼び出しから戻った後で、 Net.Data によって設定されます。変数の値は、dtw_execute() 呼び出し自体の戻り値に設定されます。この値は、その後、 Net.Data マクロの MESSAGE ブロックを処理するために Net.Data によって使用されます (この関数呼び出しについてブロックが指定された場合)。

MESSAGE ブロックを指定しない、または指定された MESSAGE ブロックの中に dtw_execute() からの戻りコードを処理するための項目がない場合、 Net.Data は、

dtw_lei_t 構造体の default_error_message フィールドの内容を表示します。このフィールドは、言語環境によって dtw_execute() ルーチンの中でいつでも設定できます。 41ページの『付録B. 言語環境のテンプレート』 のプログラム・テンプレートの setErrorMessage() ルーチンで、 default_error_message フィールドを設定する方法の例を示します。

言語環境通信構造体

Net.Data は、2 つの構造体を使用して言語環境と通信します。言語環境はこれらの構造体を処理し、この構造体内で情報の設定および受け渡しを行わなければなりません。

- dtw_lei_t
- dtw_parm_data_t

Net.Data は、言語環境インターフェース構造体 (たとえば、dtw_lei_t) を、呼び出した言語環境関数に渡します。この構造体には、とりわけ、言語環境関数に渡されるパラメーターのリストを含むパラメーター・データ配列が入っています。 Net.Data が呼び出す言語環境関数は、要求を処理し、パラメーター・データ配列内のパラメーター (該当する場合) を更新し、Net.Data に戻ります。

Net.Data は、次にパラメーター・データ配列を調べて、言語環境関数によって設定された新しい値を反映するようにパラメーターのコピーを更新し、 Net.Data マクロの処理を続行します。

dtw_lei_t 構造体

各言語環境のインターフェース関数は、dtw_lei_t 構造体のポインターを受け取ります。 dtw_lei_t 構造体の形式は次のとおりです。

```
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;
```

dtw_lei_t 構造体内のフィールド:

function_name
function_name フィールドには、関数ブロックの名前が入った文字列へのポインターが含まれています。これは、言語環境が表示するエラー・メッセージで FUNCTION ブロック名を指定するのに役立つ場合があります。

flags
flags フィールドは、Net.Data と言語環境間の通信に使用されます。以下の定数を使用して OR 演算を行って、flags フィールド・ポインターを指定します。

- Net.Data は DTW_STMT_EXEC を設定して、 exec_statement フィールドに EXEC ステートメントからのファイル名とパラメーターが含まれていることを、 dtw_execute() インターフェース関数に知らせます。
- DTW_END_ABNORMAL が Net.Data によって設定され、異常もしくは予期しない状態が発生したことと、 Net.Data の終了前に言語環境が何らかのクリーンアップ処理 (すなわち保留資源の解放) を行う必要があることを、dtw_cleanup() インターフェース関数に知らせます。
- DTW_LE_FATAL_ERROR が言語環境インターフェース関数によって設定され、言語環境に致命的エラーが発生したことを Net.Data に知らせます。このフラグが設定されると、Net.Data は Net.Data マクロの処理を停止し、フラグを DTW_END_ABNORMAL に設定して、活動中のすべての言語環境の dtw_cleanup() インターフェース関数を呼び出し、デフォルト・メッセージを印刷して、終了します。フラグは、言語環境呼び出しで非ゼロの戻り値が戻された場合にのみチェックされます。
- DTW_LE_MSG_KEEP が言語環境インターフェース関数によって設定され、default_error_message が指し示す記憶域を解放してはならないことを、Net.Data に知らせます。この定数が設定されないと、Net.Data は記憶域を解放しようとします。
- DTW_LE_CONTINUE が dtw_execute() インターフェース関数によって設定され、 Net.Data に dtw_getNextRow() インターフェース関数を呼び出すよう指示します。 Net.Data が dtw_getNextRow() を呼び出すのは、フラグが設定され、呼び出しから dtw_execute() インターフェース関数への戻り値がゼロの場合のみです。

exec_statement

exec_statement フィールドには、以下のいずれかのポインターが入っています。

- FUNCTION ブロックから、(変数置換後の) 実行可能ステートメントを含むストリングへのポインター
- EXEC ステートメントからファイル名およびパラメーターへのポインター

parm_data_array

parm_data_array フィールドには、dtw_parm_data_t 構造体の配列のポインターが入ります。配列は、ゼロの入った parm_data 構造体で終わります。dtw_parm_data_t 構造体は、変数や関連する値を言語環境に渡したり、言語環境によって行われる可能性がある変数値の変更を取得したりする場合に、Net.Data によって使用されます。構造体については、6ページの『dtw_parm_data_t 構造体』 を参照してください。

default_error_message

言語環境によって、エラー条件を記述する文字ストリングに default_error_message フィールドが設定されます。呼び出しから言語環境インターフェース関数への戻り値が非ゼロで、戻り値が MESSAGE ブロックのメッセージの値と一致しない場合は、デフォルト・メッセージが表示されます。それ以外の場合、Net.Data は MESSAGE ブロックから選択されたメッセージを表示します。

le_opaque_data

le_opaque_data フィールドは、言語環境のインターフェース関数のいずれによ

っても設定されて、インターフェース関数間でパラメーターを受け渡します。 Net.Data は、ポインターを保管し、それを Net.Data が呼び出す別のインターフェース関数に渡します。 Net.Data マクロの処理後で、かつNet.Data の呼び出し側に戻る前に、 Net.Data はそのポインターを NULL に定義します。フィールドは、スレッドに固有であるため、言語環境はスレッド固有のデータを保管することができます。このフィールドを使用するのは、dtw_cleanup() インターフェース関数を持っていて、その関数が le_opaque_data フィールドに関連する記憶域を解放できる場合に限ります。

row
row フィールドは、言語環境の dtw_getNextRow() インターフェース関数を呼び出す前に、行オブジェクトに対して Net.Data によって設定されます。dtw_getNextRow() 関数は、Net.Data 行ユーティリティー・インターフェース関数を使用して、表データの行をオブジェクトに挿入します。その後、Net.Data はその行を処理し、処理する行がなくなるまで dtw_getNextRow() を呼び出します。

`reserved` フィールドは、IBM 専用の予約フィールドです。

dtw_parm_data_t 構造体

Net.Data は、dtw_parm_data_t 構造体を使用してパラメーターを言語環境に渡します。パラメーターは、次の 3 つのソースから得られます。

- FUNCTION ブロック定義で指定された明示的なパラメーター
- FUNCTION ブロック定義の RETURNS キーワードで指定された戻り変数
- Net.Data 初期設定ファイルの ENVIRONMENT 構成ステートメントで指定されたパラメーター

Net.Data は、最初に明示的パラメーター、続いて ENVIRONMENT ステートメントで指定されたパラメーター、次に戻り変数を渡します。

dtw_parm_data_t 構造体の形式は次のとおりです。

```
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;
```

dtw_parm_data_t 構造体内のフィールド:

parm_descriptor
parm_descriptor フィールドは、言語環境に渡されるパラメーターの型と使用方法を記述します。 Net.Data は、次の定数を使用して OR 演算を実行することによってフィールドを設定します。

- DTW_IN は、パラメーターが入力専用パラメーターであることを示します。
- DTW_OUT は、パラメーターが出力専用パラメーターであることを示します。
- DTW_INOUT は、パラメーターが入出力パラメーターであることを示します。

- DTW_STRING は、パラメーター値がストリングを指すポインターであることを示します。
- DTW_TABLE は、パラメーター値が表を指すポインターであることを示します。

  Net.Data は、parm_descriptor フィールドを、常に DTW_IN、DTW_OUT、または DTW_INOUT に設定し、 DTW_STRING および DTW_TABLE との論理 OR を使用します。

parm_name

parm_name フィールドは、パラメーターの名前を含む文字列へのポインターです。パラメーターがリテラル・ストリングの場合、Net.Data は、このポインターを NULL に設定します。

parm_value

parm_value フィールドは、パラメーターの値を含むオブジェクトへのポインターです。パラメーターがまだ定義されていない変数である場合、このポインターは、Net.Data によって NULL に設定されます。

res1 および res2 フィールドは予約フィールドです。

parm_name および parm_value はともに、Net.Data 実行時 *heap*、すなわち Net.Data が動的メモリー割り当てに使用するメモリーのエリアから割り当てられたオブジェクトを指し示します。 parm_name または parm_value が別のストリングに置き換えられると、元のストリングは解放され、Net.Data ヒープから割り当てられた文字ストリングを指し示すポインターに置き換えられなければなりません。元のストリングの解放には、dtw_malloc() および dtw_free() ユーティリティー関数を使用します。

言語環境インターフェース関数

Net.Data は 4 つのインターフェース関数を言語環境に使用します。ユーザーはこれらの関数のうちの 1 つまたは複数を提供します。これらの関数のうち 3 つは任意選択ですが、それぞれの言語環境に dtw_execute() インターフェース関数がなければなりません。 Net.Data マクロが dtw_execute() インターフェース関数を持たない言語環境を参照すると、 Net.Data はエラー・メッセージを戻し、Net.Data マクロの処理を停止します。

言語環境を呼び出すには、それを Net.Data マクロの FUNCTION ブロックで参照します。言語環境インターフェース関数は、以下の順序で呼び出されなければなりません。

1. dtw_initialize()
2. dtw_execute()
3. dtw_getNextRow()
4. dtw_cleanup()

dtw_execute() 関数は、言語環境で指定しなければならない唯一のインターフェース関数です。

Net.Data は、言語環境を使用する関数の呼び出しを検出すると、以下のステップに従ってその言語環境を呼び出します。

1. Net.Data は、dtw_initialize() がこの言語環境に定義されている場合はこれを呼び出します。この関数は、データベースへの接続や変数の割り当てなどの、言語環境が必要とするすべての初期化タスクを実行します。
2. Net.Data は、dtw_execute() を呼び出して、言語環境が処理しなければならないステートメントが入ったマクロ FUNCTION ブロックを処理します。
3. Net.Data は、dtw_execute() が、正常に戻ったときに dtw_getNextRow() を呼び出す必要があることを示した場合は、dtw_getNextRow() を呼び出します。
4. Net.Data マクロ処理が完了すると、この関数が言語環境に定義されている場合には、 Net.Data は dtw_cleanup() を呼び出して環境のクリーンアップ処理 (たとえば、データベースからの切断や変数の解放) を行い、その後で Webサーバーに戻ります。

インターフェース関数は、以下の節で説明します。

- 『dtw_initialize()』
- 『dtw_execute()』
- 9ページの『dtw_getNextRow()』
- 10ページの『dtw_cleanup()』

dtw_initialize()

dtw_initialize() インターフェース関数は、データベースへの接続や変数の割り当てなどの、言語環境が必要とするすべての特殊な初期化を行います。このインターフェース関数は、1 度呼び出され、かつ任意選択です。

Net.Data は、言語環境を参照する FUNCTION ブロックを最初に呼び出すときに、マクロごとに言語環境の dtw_initialize() インターフェース関数を 1 度だけ呼び出します。次の言語環境への参照では、dtw_initialize() インターフェース関数の呼び出しはバイパスされます。

このインターフェース関数は、メッセージ・ブロック処理には影響を与えません。正またはゼロの戻りコードは処理が継続されることを意味し、負の戻りコードは処理が継続されないことを意味します。戻りコードが非ゼロであり、default_error_message フィールド内でデフォルト・メッセージが定義されている場合は、デフォルト・メッセージが出されます。デフォルト・メッセージがない場合は、Net.Data によってエラー・メッセージが出されます。

dtw_execute()

dtw_execute() インターフェース関数は、言語環境で処理しなければならないステートメントが入ったマクロ FUNCTION ブロックを処理します。たとえば、データベース言語環境を参照する FUNCTION ブロックには、言語環境がデータベースを照会する際に使用する SQL ステートメントが入っています。

dtw_execute() インターフェース関数は、言語環境を参照する FUNCTION ブロックを Net.Data マクロが処理するときは必ず呼び出されます。 dtw_execute() インターフェース関数が完了したときに次に何が起こるかは、言語環境が表データを一度に 1 行ずつ処理するかどうかによって決まります。 1 行ずつ処理する場合、インターフェース

関数は DTW_LE_CONTINUE フラグを dtw_lei_t 構造体に設定して、 Net.Data に dtw_getNextRow() インターフェース関数を呼び出すように指示します。dtw_getNextRow() インターフェース関数とその処理ステップの詳細については、『dtw_getNextRow()』 を参照してください。

dtw_execute() インターフェース関数に、レポート・ブロック処理用の入力データの作成に必要なすべての処理を行わせると、パフォーマンスを最適化することができます。たとえば、dtw_execute インターフェース関数は、レポート・ブロック・フェーズの間に処理する表全体を生成することができます。

dtw_getNextRow()

dtw_getNextRow() インターフェース関数は、Net.Data 表を一度に 1 行処理するための入力データを検索します。この関数は、この表には別の行のデータを処理する必要があることを示す、 DTW_LE_CONTINUE フラグが設定されるたびに呼び出されます。 dtw_getNextRow() はデータベース言語環境に使用します。

制約事項: このインターフェース関数が呼び出されるのは、 Net.Data が OS/400 または OS/390 オペレーティング・システム上で稼働している場合に限られます。

Net.Data は、以下の条件が満たされると dtw_getNextRow() を呼び出します。

- 言語環境の dtw_execute() 呼び出しの呼び出しが正常に完了する (戻り値がゼロ)
- dtw_execute() インターフェース関数が dtw_lei_t 構造体に DTW_LE_CONTINUE フラグを設定している

dtw_execute() 関数が DTW_LE_CONTINUE フラグをオンに設定すると、 Net.Data は以下のステップを実行します。

1. dtw_execute() インターフェース関数の戻り値のメッセージ・ブロックを処理します。
2. 言語環境の dtw_getNextRow() インターフェース関数を呼び出し、一度に 1 行処理を始めます。
3. レポート・ブロックを処理します。
4. dtw_getNextRow() インターフェース関数の戻り値のメッセージ・ブロックを処理します。
5. dtw_getNextRow() が DTW_LE_CONTINUE フラグをオンにしているかどうかを判別します。
   - オンならば、ステップ 2 の dtw_getNextRow() インターフェース関数の処理を続行します。
   - オンでなければ、一度に 1 行処理は終了し、Net.Data は Net.Data マクロの処理を続けます。

dtw_getNextRow() が呼び出されると、dtw_lei_t 構造体の row フィールドは、行オブジェクトを指すように設定されます。行オブジェクトを操作するには、 Net.Data ユーティリティー関数の dtw_row_SetCols() および dtw_row_SetV() を使用します。Net.Data は、dtw_getNextRow() インターフェース関数を最初に呼び出した後は、行オブジェクトにその表の列見出しが入っているものと見なします。 後続の呼び出しでは、実際の表データが入ります。

dtw_getNextRow() 関数の呼び出しは、 (メッセージ・ブロック処理でほかの指示が行われない限り) DTW_LE_CONTINUE フラグが設定されている間続けられます。

dtw_cleanup()

dtw_cleanup() インターフェース関数は、dtw_initialize() を用いて言語環境を初期化する場合に、言語環境をクリーンアップ処理するのに使用します。この関数は、データベースから切断したり、変数を解放したりするようなタスクの場合に使用します。このインターフェース関数は、任意選択です。

Net.Data 要求を処理している間、Net.Data は、Net.Data の処理が終了した場合、またはエラーで Net.Data がマクロの処理を停止した場合に、言語環境の dtw_cleanup() インターフェース関数を 1 回呼び出します。

Net.Data は、クリーンアップ処理に異常条件が生じると、 dtw_lei_t 構造体の flags フィールドを DTW_END_ABNORMAL に設定します。以下の異常条件で、dtw_cleanup() を使用する場合の例を示します。

- 言語環境インターフェース関数が、 dtw_lei_t 構造体の flags フィールドで DTW_LE_FATAL_ERROR ビットを設定して、致命的エラーが発生したことを示している。
- Net.Data が回復不能エラーを検出した。
- Net.Data マクロ・メッセージ・ブロック処理の結果終了した。

言語環境のインターフェース関数が、インターフェース関数間で渡されるパラメーターを使用して le_opaque_data フィールドを設定する場合は、処理の終了時に dtw_cleanup() を使用してフィールドを解放してください。

このインターフェース関数は、メッセージ・ブロック処理には影響を与えません。戻り値が非ゼロで、デフォルト・メッセージが出される場合で、デフォルト・メッセージがない場合は、マクロ処理プログラムが警告メッセージを出します。

言語環境ステートメントの設計

各言語環境ごとに、その言語環境に固有の情報を含む Net.Data 初期設定ファイル内に ENVIRONMENT ステートメントがあります。新しい言語環境を作成するときは、初期設定ファイル用の環境ステートメントを設計し、それをユーザーが初期設定ファイルに追加する方法を文書化する必要があります。

ENVIRONMENT ステートメントは、Net.Data が呼び出す必要がある言語環境に関する情報を指定し、言語環境名などの言語環境 DLL または共用ライブラリー、 DLL または共用ライブラリー名、および関数呼び出しごとに言語環境に渡されるパラメーターのリストをロードします。

Net.Data は、呼び出されたときに構成情報を読み取りますが、その言語環境を識別する FUNCTION ブロックがマクロ内から呼び出されるまでは、言語環境 DLL または共用ライブラリーのロードは行いません。 DLL は、Net.Data が終了するまでロードされたままになります。

以下の節では、構文、パラメーターの記述、およびユーザーの資料として使用可能な例について記載します。

ENVIRONMENT ステートメントの構文

ENVIRONMENT ステートメントの形式を、次に示します。

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

各 ENVIRONMENT ステートメントは、単一の行にしなければなりません。

以下は、言語環境ごとに指定しなければならないパラメーターです。

- *type*

  この言語環境を Net.Data マクロの FUNCTION ブロック定義に関連付けるための名前。 FUNCTION ブロック定義で言語環境の型も指定して、関数呼び出しを処理する言語環境を Net.Data に知らせる必要があります。FUNCTION ブロックの詳細については、*Net.Data 解説書* の「Function ブロック」の節を参照してください。

  重要: 名前の接頭部を `DTW` にすることはできません。この接頭部は、Net.Data に付属の言語環境用に予約されています。 DTW 接頭部を使用すると、Net.Data は言語環境の DLL をロードすることができません。

- *library_name*

  Net.Data が呼び出す言語環境インターフェースが入っている、オブジェクトの名前。ファイル拡張子は、オペレーティング・システムごとに異なります。

  – AIX® では、共用ライブラリーの名前は、拡張子 *.o* を付けて指定します。

  – HP/UX では、共用ライブラリーの名前は、拡張子 *.sl* を付けて指定します。

  – OS/2® および Windows NT では、 DLL 名は、拡張子 *.dll* を付けて指定します。

  – OS/390® では、DLL 名は、拡張子 *.dll* を付けて指定します。

  – OS/400® では、サービス名プログラム名は、拡張子 *.SRVPGM* を付けて指定します。

  SUN、SCO、および LINUX では、共用ライブラリーの名前は、拡張子 *.so* を付けて指定します。

  この名前の指定方法を調べる場合は、それぞれのオペレーティング・システム対応の Net.Data と共に出荷される初期設定ファイルを参照してください。完全修飾パス名を使用して、Net.Data が DLL または共用ライブラリーを検索できるようにすることを考慮してください。

- *specification*

  Net.Data がそのパラメーターを入力、出力、または入出力のいずれに使用するかを示す指定を渡すパラメーター。有効な値は次のとおりです。

  IN 入力に使用するパラメーター。

  OUT 出力に使用するパラメーター。

  INOUT 入出力両方に使用するパラメーター。

- *parameter_list*

  FUNCTION ブロック定義で指定されたパラメーターに加えて、関数呼び出しごとに言語環境に渡されるパラメーターのリスト。これらのパラメーターは、 FUNCTION

ブロック定義で指定されたパラメーターに続いて、 *dtw_lei_t* 構造体の *parm_data_array* フィールドで渡されます。関数を呼び出す前に、 Net.Data マクロでこれらのパラメーターを変数として定義しなければなりません。関数がこれらのパラメーターの値を変更した場合、関数が処理を終了すると、パラメーターは変更された値を維持します。

ENVIRONMENT ステートメントの例

以下の例で、Net.Data が提供する言語環境の ENVIRONMENT ステートメントを示します。これらの例で、パラメーターの指定方法を説明します。 ENVIRONMENT ステートメントに組み込む変数は、Net.Data マクロ書き込みプロセスに、そのマクロで定義もしくはオーバーライドできるようにさせたい変数です。その他の例については、 *Net.Data 解説書* の付録のオペレーティング・システム固有の情報、もしくは Net.Data README ファイルまたはプログラム資料説明書を参照してください。

次の例で、OS/2、AIX、および Windows NT の構文を使用した、 Net.Data 提供言語の ENVIRONMENT ステートメントを示します。

```
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 ステートメントは、それぞれのオペレーティング・システムで変わることがあります。たとえば、OS/390 は、SQL およびODBC アクセスの場合と若干異なります。

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

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

第2章 言語環境のプログラミング・インターフェース・ユーティリティー関数

Net.Data には、新しい言語環境を設計する際に使用するプログラミング・インターフェースが備わっています。言語環境インターフェースには、メモリーや構成変数を管理し、表および行操作フィーチャーを提供する Net.Data サービスにアクセスするユーティリティー関数があります。 41ページの『付録B. 言語環境のテンプレート』に、言語環境を設計する際にモデルとして使用できるテンプレートを記載します。

以下の節で、Net.Data 言語環境インターフェース・ユーティリティー関数を説明します。

言語環境ユーティリティー関数

言語環境は、ユーティリティー関数を用いて Net.Data サービスにアクセスします。ユーティリティー関数は、次の 4 つに区分されます。

- 『メモリー管理用のユーティリティー関数』
- 14ページの『構成変数を管理するためのユーティリティー関数』
- 14ページの『表操作用のユーティリティー関数』
- 15ページの『行操作用のユーティリティー関数』

メモリー管理用のユーティリティー関数

言語環境は、メモリー管理ユーティリティー関数を用いて、Net.Data が所有する記憶域を割り当て、 Net.Data 実行時ライブラリーを用いて割り当てた記憶域を解放します。

以下の例で、これらのユーティリティー関数の必要性を説明します。 Net.Data が、コンパイラー A とその対応する実行時ライブラリーを用いて作成されているものとします。プログラマーが、新しい言語環境を作成しますが、実行時ライブラリーが異なるコンパイラー B を使用するものとします。 2 つの実行時ライブラリーの間に互換性がない可能性があるため、言語環境は Net.Data が割り当てた記憶域を解放できず、 Net.Data は言語環境が割り当てた記憶域を解放できません。

表 1. メモリー管理ユーティリティー関数

ユーティリティー関数	説明
18ページの『dtw_malloc()』	dtw_malloc() を用いて Net.Data の実行時ヒープから記憶域を割り当てます。
16ページの『dtw_free()』	dtw_malloc() を用いて Net.Data の実行時ヒープから割り当てられた記憶域を解放します。
21ページの『dtw_strdup()』	dtw_malloc() を用いて、Net.Data の実行時ヒープから記憶域を割り当て、指定されたストリングを、割り当てた記憶域にコピーします。

© Copyright IBM Corp. 1997, 1998

構成変数を管理するためのユーティリティー関数

構成変数用の管理ユーティリティー関数を用いると、言語環境は、Net.Data 初期設定ファイルに格納されている構成情報にアクセスすることができます。すべての言語環境は、これらの関数を用いて Net.Data 初期設定ファイルを共用し、そのファイル内の情報を言語環境の構成に使用することができます。

表 2. 構成ユーティリティー関数

ユーティリティー関数	説明
17ページの『dtw_getvar()』	Net.Data 初期設定ファイルから構成変数の値を取り出します。

表操作用のユーティリティー関数

表関数は、言語環境に渡される任意の Net.Data マクロの表変数を操作するのに使用します。

行および列番号は 1 から始まります。

表 3. 表ユーティリティー関数

ユーティリティー関数	説明
32ページの『dtw_table_New()』	表オブジェクトを作成します。
24ページの『dtw_table_Delete()』	表オブジェクトを削除します。
35ページの『dtw_table_SetCols()』	表の幅を設定し、列見出し用の記憶域を割り当てます。
28ページの『dtw_table_GetV()』	表値を検索します。
37ページの『dtw_table_SetV()』	表値を設定します。
27ページの『dtw_table_GetN()』	表の列見出しを取り出します。
36ページの『dtw_table_SetN()』	表の列見出しを設定します。
34ページの『dtw_table_Rows()』	表内の現在の行数を検索します。
23ページの『dtw_table_Cols()』	表内の現在の列数を検索します。
31ページの『dtw_table_MaxRows()』	表内の最大許容行数を検索します。
33ページの『dtw_table_QueryColnoNj()』	列の列番号を取り出します。
22ページの『dtw_table_AppendRow()』	表の終わりに 1 つまたは複数の行を追加します。
30ページの『dtw_table_InsertRow()』	表に 1 つまたは複数の行を挿入します。
26ページの『dtw_table_DeleteRow()』	表から 1 つまたは複数の行を削除します。

表 3. 表ユーティリティー関数 (続き)

ユーティリティー関数	説明
29ページの『dtw_table_InsertCol()』	表に 1 つまたは複数の列を挿入します。
25ページの『dtw_table_DeleteCol()』	表から 1 つまたは複数の列を削除します。

行操作用のユーティリティー関数

行ユーティリティー関数は、一度に 1 行を処理する間に言語環境の dtw_getNextRow() インターフェース関数に渡される、行オブジェクトを操作します。

行番号は 1 から始まります。

表 4. 行ユーティリティー関数

ユーティリティー関数	説明
19ページの『dtw_row_SetCols()』	行の幅を設定します。
20ページの『dtw_row_SetV()』	表値を設定します。

ユーティリティー関数の構文参照

この節では、ユーティリティー関数、その形式、使用法、およびパラメーターのそれぞれを説明するとともに、簡単な例を記載します。

dtw_free()

使用法

dtw_malloc() を用いて Net.Data の実行時ヒープから割り当てた記憶域を解放します。buffer は、解放する割り当て済み記憶域を指します。

形式

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

パラメーター

buffer	解放する割り当て済み記憶域へのポインター。

例

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

myBuf = (char *)dtw_malloc(nbytes);

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

dtw_getvar()

使用法

Net.Data 初期設定ファイルから *var_name* が指定した構成変数の値を取り出します。Net.Data は、dtw_getvar() から戻されたメモリーを所有しますが、その変更も解放も行いません。

形式

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

パラメーター

var_name	取り出す構成変数の名前。

例

```
char *myBindFile;

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

dtw_malloc()

使用法

dtw_malloc() を用いて、Net.Data の実行時ヒープから割り当てた記憶域を指し示すポインターを戻します。記憶域の長さは、*nbytes* です。 Net.Data は、要求された記憶域を戻せない場合は、NULL ポインターを戻します。

形式

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

パラメーター

nbytes	割り当てるバイトの数。

例

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

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

dtw_row_SetCols()

使用法

行の幅を割り当て、列見出しの記憶域を割り当てます。 dtw_row_SetCols() ユーティリティー関数は、行ごとに 1 回使用することができます。

形式

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

パラメーター

row	まだ列が割り当てられていない、新たに作成された行へのポインター。
cols	新しい行に割り当てる列数の初期値。

例

```
void *myRow;

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

dtw_row_SetV()

使用法

表値を割り当てます。 dtw_row_SetV() ユーティリティー関数の呼び出し側は、*src* が指し示すメモリーの所有権を保存します。現行表値を削除する場合は、この値を NULL に割り当てます。

形式

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

パラメーター

row	修正する行へのポインター。
src	設定する新しい値が入っている文字列。
col	設定する値の列番号。

例

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

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

dtw_strdup()

使用法

dtw_malloc() を用いて、Net.Data の実行時ヒープから記憶域を割り当て、*string* により指定されたストリングを、割り当てた記憶域にコピーします。 Net.Data は、要求された記憶域を戻せない場合は、NULL ポインターを戻します。

形式

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

パラメーター

string 割り当て済み記憶域に複写する文字列値へのポインター。

例

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

myDupString = dtw_strdup(myString);
```

dtw_table_AppendRow()

使用法

表の終わりに 1 つまたは複数の行を追加します。表に数行追加した後、dtw_table_SetV() ユーティリティーで新しい行の表値を割り当てます。

形式

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

パラメーター

table	追加が行われる表へのポインター。
rows	追加される行の数。

例

```
void *myTable;

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

dtw_table_Cols()

使用法

表内の現在の列数を戻します。

形式

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

パラメーター

table 現在の列数が戻される表へのポインター。

例

```
void *myTable;
int currentColumns;

currentColumns = dtw_table_Cols(myTable);
```

dtw_table_Delete()

使用法

列見出し、表値、および表オブジェクトのすべてを削除します。

形式

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

パラメーター

table 削除する表へのポインター。

例

```
void *myTable;

rc = dtw_table_Delete(myTable);
```

dtw_table_DeleteCol()

使用法

start_col で指定された列から始まる 1 つまたは複数の列を削除します。表の行と列をすべて削除する場合は、*cols* パラメーターに代えてユーティリティー関数 dtw_table_Cols() を使用します。

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

形式

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

パラメーター

table	修正する表へのポインター。
start_col	削除する最初の列の列番号。
rows	削除する列の数。

例

```
void *myTable;

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

dtw_table_DeleteRow()

使用法

*start_row*で指定された列から始まる 1 つまたは複数の行を削除します。

形式

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

パラメーター

table	修正する表へのポインター。
start_row	削除する最初の行の行番号。
rows	削除する行の数。

例

```
void *myTable;

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

dtw_table_GetN()

使用法

列見出しを取り出します。 Net.Data は、*dest* が指し示すメモリーを所有しますが、その変更も解放も行いません。

形式

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

パラメーター

table	列見出しが取り出される表へのポインター。
dest	列見出しが入る文字列へのポインター。
col	列見出しの列番号。

例

```
void *myTable;
char *myColumnHeading;

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

dtw_table_GetV()

使用法

表から値を取り出します。 Net.Data は、*dest* が指し示すメモリーを所有しますが、その変更も解放も行いません。

形式

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

パラメーター

table	値が取り出される表へのポインター。
dest	値を含む文字ストリングを指し示すポインター。
row	取り出す値の行番号。
col	取り出す値の列番号。

例

```
void *myTable;
char *myTableValue;

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

dtw_table_InsertCol()

使用法

指定された列の後に 1 つまたは複数の列を挿入します。

形式

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

パラメーター

table	修正する表へのポインター。
after_col	新しい列が挿入される列の番号 (この列の後に挿入されます)。この表の始めに列を挿入する場合は、0 を指定します。
cols	挿入する列の数。

例

```
void *myTable;

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

dtw_table_InsertRow()

使用法

指定された行の後に 1 つまたは複数の行を挿入します。

形式

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

パラメーター

table	修正する表へのポインター。
after_row	新しい行が挿入される行の番号 (この行の後に挿入されます)。この表の始めに行を挿入する場合は、0 を指定します。
rows	挿入する行の数。

例

```
void *myTable;

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

dtw_table_MaxRows()

使用法

dtw_table_New() ユーティリティー関数パラメーター *row_lim*で定義されると、Net.Data 表に許可された行の最大数を戻します。

形式

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

パラメーター

table 行の最大数が戻される元の表を指し示すポインター。

例

```
void *myTable;
int maximumRows;

maximumRows = dtw_table_MaxRows(myTable);
```

dtw_table_New()

使用法

Net.Data 表オブジェクトを作成し、すべての列見出しおよびフィールド値を NULL に初期化します。呼び出し側が、行数と列数の初期値および行の最大数を指定します。行および列の初期数が 0 の場合は、表関数呼び出しの前に、dtw_table_SetCols() 関数を用いて行内のフィールド数を指定しなければなりません。

形式

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

パラメーター

table	新しい表の名前。
rows	新しい表に割り当てる行数の初期値。
cols	新しい表に割り当てる列の初期数。
row_lim	この表に含めることができる行の最大数。

例

```
void *myTable;

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

dtw_table_QueryColnoNj()

使用法

列見出しに関連する列番号を戻します。

形式

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

パラメーター

table	照会する表へのポインター。
name	列番号が戻される列見出しを指定する文字列。その列見出しが表に存在しない場合には、0 が戻されます。

例

```
void *myTable;
int columnNumber;

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

dtw_table_Rows()

使用法

表内の現在の行数を戻します。

形式

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

パラメーター

table 現在の行数が戻される表へのポインター。

例

```
void *myTable;
     int currentRows;

     currentRows = dtw_table_Rows(myTable);
```

dtw_table_SetCols()

使用法

表の列数を設定し、列見出し用の記憶域を割り当てます。列見出しを表の作成時に指定します。表の作成時に指定しない場合は、他の表関数を使用する前に、このユーティリティー関数を呼び出して指定しなければなりません。 dtw_table_SetCols() ユーティリティー関数を使用できるのは、1 つの表につき 1 回だけです。その後は、dtw_table_DeleteCol() ユーティリティー関数または dtw_table_InsertCol() ユーティリティー関数を使用してください。

形式

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

パラメーター

table	列または行が割り当てられていない新しい表へのポインター。
cols	新しい表に割り当てる列の初期数。

例

```
void *myTable;

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

dtw_table_SetN()

使用法

列見出しに名前を割り当てます。 dtw_table_SetN() ユーティリティー関数の呼び出し側が、*src*パラメーターが指し示すメモリーの所有権を保存します。列見出しを削除する場合は、列見出し値を NULL に割り当てます。

形式

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

パラメーター

table	列見出しが割り当てられる表を指し示すポインター。
src	新しい列見出しに割り当てられる文字ストリング。
col	列の番号。

例

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

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

dtw_table_SetV()

使用法

表内の値を割り当てます。 dtw_table_SetV() ユーティリティー関数の呼び出し側が、*src*パラメーターが指し示すメモリーの所有権を保存します。表値を削除する場合は、この値を NULL に割り当てます。

形式

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

パラメーター

table	値が割り当てられる表を指し示すポインター。
src	新しい値に割り当てられる文字ストリング。
row	新しい値の行番号。
col	新しい値の列番号。

例

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

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

付録A. Net.Data テクニカル・ライブラリー

Net.Data テクニカル・ライブラリーは、次の Net.Data の Web サイトから入手できます。

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

マニュアル名	説明
• *Net.Data 管理およびプログラミングの手引き OS/390 版* • *Net.Data 管理およびプログラミングの手引き OS/2、Windows NT、および UNIX 版* • *Net.Data 管理およびプログラミングの手引き OS/400 版*	Net.Data のインストール、構成、および起動に関する概念とタスクに関する内容です。また、Net.Data マクロの作成方法、Net.Data パフォーマンス・テクニックの使い方、 Net.Data 言語環境の使い方、接続の管理方法、およびトラブルシューティングとパフォーマンス調整のための Net.Data ロギングとトレースの使い方も説明されています。
Net.Data 解説書	Net.Data マクロ言語、変数、および組み込み関数について説明されています。
Net.Data 言語環境解説書	Net.Data 言語環境インターフェースについて説明されています。
Net.Data メッセージ	Net.Data エラー・メッセージと戻りコードが一覧されています。

© Copyright IBM Corp. 1997, 1998

付録B. 言語環境のテンプレート

このテンプレートは、独自の言語環境を作成する場合に使用します。

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

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

図 2. 言語環境のテンプレート (1/14)

© Copyright IBM Corp. 1997, 1998

```
#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
```

図 2. 言語環境のテンプレート (2/14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* 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;
}
```

図 2. 言語環境のテンプレート (3/14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* 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)
          {
          }
      }
```

図 2. 言語環境のテンプレート (4/14)

```
    /*-----------------------------------------------------------------*/
    /* 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;
}
```

図 2. 言語環境のテンプレート (5/14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* 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;
}
```

図 2. 言語環境のテンプレート (6/14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* 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;
}
```

図 2. 言語環境のテンプレート (7/14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* 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) {
```

図 2. 言語環境のテンプレート (8/14)

```
        /*-----------------------------------------------------------*/
        /* 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;
}
```

図 2. 言語環境のテンプレート (9/14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* 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. */
    /*---------------------------------------------------------------*/
```

図 2. 言語環境のテンプレート (10/14)

```
/    /*-----------------------------------------------------------------*/
    /* 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;
```

図 2. 言語環境のテンプレート (11/14)

```
/*-------------------------------------------------*/
/* 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);
      }
  }
```

図 2. 言語環境のテンプレート (12/14)

```
                    /*-----------------------------------------------*/
                    /* 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;
}
```

図 2. 言語環境のテンプレート (13/14)

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* 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;
}
```

図 2. 言語環境のテンプレート (14/14)

付録C. ファイル例の作成

次の例は、各種のオペレーティング・システムで LE を作成する場合の参考です。オペレーティング・システムおよび環境によって独特の特性があるので、これらの例は単なる参考とお考えください。共用ライブラリーを作成する場合は、オペレーティング・システムで使用しているコンパイラーに付属のマニュアルを参照してください。

OS/390 JCL の例

```
//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)
```

© Copyright IBM Corp. 1997, 1998

```
//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)
//*
```

makefile の例 (OS/390 固有)

```
### 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)
```

OS/400 CL の例

次の条件を前提として、その後に続くステップに従い、AS/400 上で言語環境ファイルを作成してください。

- SRC はソース・ファイルです (C 言語)。
- MYLE には、エクスポート可能なプロシージャー `dtw_execute` が入っています。
- ファイル QSRVSRC、メンバー MYLEEXP、には、プロシージャー `dtw_execute` をエクスポートするための指定が入っています。

1. モジュールを作成します。

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

2. サービス・プログラムを作成します。

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

QTCP/QTMHLE は、すべての Net.Data バインド可能 API が入っているサービス・プログラムです。 V4R3 以降のリリースでは、QTCP/QTMHLE を利用できますが、QHTTPSVR/QTMJLE を使用するようにしてください。

付録D. 特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、 IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM によって明示的に指示されたものを除き、これらのプログラムまたは製品に関連する動作の評価および検証はお客様の責任で行っていただきます。

IBM および他社は、本書で説明する主題に関する特許権 (特許出願を含む) 商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木3丁目2-31
AP事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

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

本プログラムに関する上記の情報は、適切な条件の下で、使用することができますが、有償の場合もあります。

本書において解説されているライセンス・プログラムおよびそのライセンス・プログラム資料は、「IBM プログラム使用契約書」および「IBM プログラムのご使用条件」の契約条件に基づいて弊社から提供されるものです。

IBM 以外の製品に関する情報は、それぞれの製品の提供元、それに関する印刷物、その他の公に使用可能な情報源から得たものです。 IBM はそれらの製品をテストしておらず、それら IBM 以外の製品に関して、パフォーマンスの正確さ、互換性、またはその他についての苦情を受け付けることはできません。 IBM 以外の製品の機能に関しては、それぞれの製品の提供元にお問い合わせください。

著作権の使用許諾:

© Copyright IBM Corp. 1997, 1998

本書には、IBM が説明するための一例として提供している簡単なプログラムが含まれています。これらの例は必ずしもすべての場合について完全にテストされたものではありません。 IBM はこれらのプログラムの信頼性、可用性、および機能について法律上の瑕疵担保責任を含むいかなる明示または暗示の保証責任も負いません。本書中に含まれているすべてのプログラムは ″現存するままの状態″ で提供されます。IBM はプログラムの商業的な使用可能性および特定の目的に対する適合性については、いかなる保証も行いません。

商標

以下の用語は、米国またはその他の国における IBM Corporation の商標です。

AIX
AS/400
CBIDO
CBPDO
CICS
CustomPac
DB2
DB2 Universal Database
DataJoiner
Distributed Relational Database Architecture
DRDA
IBM
IMS
Language Environment
MVS/ESA
Net.Data
OpenEdition
Operating System/400
OS/2
OS/390
OS/400
RACF
SystemPac

以下の用語は、米国またはその他の国における IBM Corporation の商標です。

Java およびすべての Java ベースの商標およびロゴは、米国ならびにその他の国における Sun Microsystems, Inc. の商標です。

Lotus および Domino Go Webserver は、米国ならびにその他の国における Lotus Development Corporation の商標です。

Microsoft、Windows、Windows NT、および Windows のロゴは、米国ならびにその他の国における Microsoft Corporation の商標または登録商標です。

ダブル・アスタリスク (**) が付いているその他の会社名、製品名、およびサービス名は、その他の会社の商標またはサービス・マークです。

用語集

絶対パス (absolute path). オブジェクトのフルパス名。絶対パス名の先頭は最上位のディレクトリー、すなわち「ルート」ディレクトリーである (フォワード・スラッシュ文字 (/) またはバック・スラッシュ (\) 文字で識別)。

API. アプリケーション・プログラミング・インターフェース (Application programming interface)。 Net.Data は、3 つの Web サーバー API をサポートすることで、CGI プロセスを介するパフォーマンスを向上させる。

アプレット (applet). HTML ページに組み込まれる Java プログラム。アプレットは、Netscape Navigator などの Java 対応のブラウザーで実行され、 HTML ページの処理時にロードされる。

アプリケーション・プログラミング・インターフェース (application programming interface (API)). オペレーティング・システムまたは別途発注可能なライセンス・プログラムによって提供される機能インターフェース。これにより、高水準言語で書かれたアプリケーション・プログラムが、オペレーティング・システムまたはライセンス・プログラムの特定のデータもしくは機能を使用できるようになる。 Net.Data は、次の専用 Web サーバー API をサポートすることで、 CGI プロセスを介したパフォーマンスを向上させる。 ICAPI、GWAPI、ISAPI、および NSAPI。

キャッシュ (cache). 最近アクセスしたデータが格納されるメモリーまたはディスク・スペースの一部。同じデータに続けてアクセスする場合の速度向上が目的。キャッシュは、ネットワークを介してアクセス可能な、使用頻度の高いデータのローカル・コピーを保留するのに使用される場合が多い。

キャッシング (caching). ローカルに Web サーバーに要求した結果得られる使用頻度の高いデータを高速で取り出すために、情報を最新表示するときまで保管するプロセス。

キャッシュ管理プログラム (Cache Manager). 1 つのマシン用のキャッシュを管理するプログラム。複数のキャッシュを管理できる。

CGI. コモン・ゲートウェイ・インターフェース (Common Gateway Interface)。

CLIETTE. Web サーバーからの要求を処理する、Net.Data Live コネクション内の長期実行プロセス。この接続管理プログラムは、これらの要求に対応する CLIETTE 処理のスケジュールを行う。

コミットメント制御 (commitment control). 資源についての操作が作業単位の一部である、 Net.Data が実行しているプロセス内の境界の確立。

コモン・ゲートウェイ・インターフェース (Common Gateway Interface (CGI)). Web サーバーがアプリケーション・プログラムに制御権を渡し、反対にデータを受け取る、標準的な方法。

接続管理プログラム (Connection Manager). Live コネクションのサポートに必要とされる Net.Data 内の実行可能ファイル `dtwcm`。

クッキー (cookie). HTTP サーバーによって Web ブラウザーに送信され、次にブラウザーがそのサーバーにアクセスするたびに送り返される、情報のパケット。クッキーには、サーバーが選択する任意の情報を含めることができ、特に状態がない HTTPトランザクション間で状態を保持するのに使用される。 *Free Online Dictionary of Computing* より。

現行作業ディレクトリー (current working directory). すべての相対パス名の解析基準となるプロセスの省略時ディレクトリー。

データベース (database). 表の集合、もしくは表スペースおよび索引スペースの集合。

データベース管理システム (database management system (DBMS)). データベースの作成、編成、および修正を制御し、データベース内に格納されたデータにアクセスする、ソフトウェア・システム。

データ型 (data type). 列およびリテラルの属性。

DBMS. データベース管理システム (Database management system)。

Domino *Go* Web サーバー (Domino Go Web Server). 標準接続およびセキュア接続の両方を提供する、Lotus Corp. と IBM から提供されている Web サーバー。ICAPI と GWAPI は、このサーバーに付属のインターフェースである。

ファイアウォール (firewall). 内部ネットワークを無許可の外部アクセスから保護するソフトウェアを備えたコンピューター。

フラット・ファイル・インターフェース (flat file interface). 平文ファイルのデータを読み書きすることができる、一連の Net.Data 組み込み関数。

© Copyright IBM Corp. 1997, 1998

GWAPI. *Go* Web サーバーの API。

HTML. ハイパーテキスト・マークアップ言語 (Hypertext markup language)。

HTTP. Hypertext transfer protocol (HTTP)。

ハイパーテキスト・マークアップ言語 (hypertext markup language). Web 文書の作成に使用するタグ言語。

hypertext transfer protocol (HTTP). Web サーバーとブラウザー間で使用する通信プロトコル。

ICAPI. インターネット接続 API (Internet Connection API)。 *を参照*。

インターネット (Internet). 国際的な公開されている TCP/IP コンピューター・ネットワーク。

イントラネット (Intranet). 企業ファイアウォール内の TCP/IP ネットワーク。

ISAPI. Microsoft のインターネット・サーバー API。

Java. 特にインターネット・アプリケーションに役立つ、オペレーティング・システムに影響されないオブジェクト指向プログラミング言語。

言語環境 (language environment). Net.Data マクロから、 DB2 などの外部データ・ソースや Perl などのプログラミング言語へのアクセスを行うモジュール。

Live コネクション (Live Connection). 接続管理プログラムと複数の CLIETTE からなる Net.Data コンポーネント。 Live コネクションは、データベース接続と Java 仮想マシン接続の再利用を管理する。

LOB. ラージ・オブジェクト (Large object)。

ミドルウェア (middleware). アプリケーション・プログラムとネットワークの中間にあるソフトウェア。ネットワークを介したクライアント・アプリケーション・プログラムとサーバーとの間の対話を管理する。

NSAPI. Netscape API。

ヌル (null). 情報がないことを示す特殊な値。

パス (path). ファイルを探すのに使用する検索経路。

パス名 (path name). システムにオブジェクトを見つける方法を示すこと。パス名は、オブジェクトの名前の後に一連のディレクトリー名が続く形式で表す。個々のディレクトリーとオブジェクト名は、フォワード・スラッシュ文字 (/) またはバック・スラッシュ文字 (\) で区切る。

Perl. インタープリター・プログラミング言語。

永続的 (persistence). トランザクションが複数の Net.Data 起動をスパンしている間、トランザクション全体に割り当てられている値を保持する状態。永続的にすることができる変数は 1 つだけである。また、コミットメント制御の影響を受ける資源についての操作は、明示的なコミットまたはロールバックが実行されるまで、またはトランザクションが完了するまで、活動状態のままとなる。

ポート (port). TCP/IP と高水準プロトコルもしくはアプリケーション間の通信に使用する 16 ビット数。

レジストリー (registry). ストリングを格納および取得することができるリポジトリー。

相対パス名 (relative path name). 先頭が最上位のディレクトリー、すなわち「ルート」ディレクトリーではないパス名。パス名の先頭は、プロセスの現行作業ディレクトリーとみなされる。

TCP/IP. 伝送制御プロトコル/インターネット・プロトコル (Transmission Control Protocol / Internet Protocol)。

トランザクション (transaction). 1 つの Net.Data の起動。永続的な Net.Data が使用されている場合、トランザクションは複数の Net.Data 起動をスパンすることができる。

伝送制御プロトコル/インターネット・プロトコル (Transmission Control Protocol / Internet Protocol). ローカル・エリア・ネットワークと広域ネットワークの両方に使用する、対等接続機能をサポートする一連の通信プロトコル。

URL. Uniform resource locator (URL)。

uniform resource locator (URL). HTTP サーバー、および任意選択でディレクトリーとファイル名を指名するアドレス。たとえば、`http://www.software.ibm.com/data/net.data/index.html`。

作業単位 (unit of work). 1 つの原子単位として扱われる、回復可能な一連の操作。作業単位内の操作はすべて、その操作が単一操作であるかのように、完了 (コミット) または取り消す (ロールバック) ことができる。コミットまたはロールバックできるのは、コミットメント制御の影響を受ける資源についての操作だけである。

Web サーバー (Web server). インターネット接続 (Internet Connection) などの HTTP サーバー・ソフトウェアを実行しているコンピューター。

索引

日本語, 英字, 数字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

［ア行］

異常条件
　エラー・メッセージ　5
　dtw_lei_t フラグ　5, 10
一度に 1 行処理
　dtw_getNextRow()　8, 9
　dtw_lei_t フラグ　5
　DTW_LE_CONTINUE　5
インターフェース関数
　言語環境の説明　7
　処理順序　7
　dtw_cleanup()　10
　dtw_execute()　8
　dtw_getNextRow()　9
　dtw_initialize()　8
受け渡し
　パラメーター　5
　変数　5
エラー条件　3
エラー条件メッセージ　5

［カ行］

記憶域
　解放　5, 7, 16
　割り当て　18, 19, 21, 35
　dtw_lei_t フラグ　5
行
　許容最大数の戻り　31
　現在数の検索　34
　削除　25, 26
　挿入　30
　追加　22
　幅の割り当て　19
　戻り　6, 8, 9
　dtw_getNextRow() インターフェース関数　6
行操作のユーティリティー関数　15
行の最大数　31
クリーンアップ処理
　異常条件のフラグ　5, 10
　処理後　8, 10
　dtw_lei_t フラグ　5, 10
言語環境
　インターフェース関数　7
言語環境 *(続き)*
　インターフェース・テンプレート　7
　構成　10
　構造体　4
　作成　1
　紹介　13
　初期化　7
　処理後のクリーンアップ処理　8, 10
　ステートメントの実行　8
　ユーティリティー関数　13
構成、環境の　10
構成変数
　管理用のユーティリティー関数　14
　変数値の検索　17
構造体、言語環境
　dtw_lei_t　4
　dtw_parm_data_t　6

［サ行］

指示、記憶域の　18
実行、言語環境ステートメントの　8
実行ステートメント、dtw_lei_t フラグ　4
初期化、言語環境タスクの　7, 8

［タ行］

致命的エラー、dtw_lei_t フラグ　5, 10
テンプレート、言語環境　41
動的なメモリー割り当て　7
特記事項　59

［ハ行］

パラメーター
　受け渡し　5, 6
　指定　6
　命名　7
　parm_name　7
ヒープ、Net.Data 実行時　7
表
　行の追加　22
　削除　24
　新規の作成　32
　ユーティリティー関数の操作　14
表値
　検索　28
　削除　24, 28, 37
　割り当て　20, 37
表の作成　32

© Copyright IBM Corp. 1997, 1998

変数
受け渡し　5
解放　10

［マ行］

メモリー管理ユーティリティー関数　13

［ヤ行］

ユーティリティー関数
行操作　15
言語環境　13
構成変数管理　14
表操作　14
メモリー管理　13
dtw_free()　16
dtw_getvar()　17
dtw_malloc()　18
dtw_row_SetCols()　19
dtw_row_SetV()　20
dtw_strdup()　21
dtw_table_AppendRow()　22
dtw_table_Cols()　23
dtw_table_DeleteCol()　25
dtw_table_DeleteRow()　26
dtw_table_Delete()　24
dtw_table_GetN()　27
dtw_table_GetV()　28
dtw_table_InsertCol()　29
dtw_table_InsertRow()　30
dtw_table_MaxRows()　31
dtw_table_New()　32
dtw_table_QueryColnoNj()　33
dtw_table_Rows()　34
dtw_table_SetCols()　35
dtw_table_SetN()　36
dtw_table_SetV()　37
用語集　60

［ラ行］

列
削除　25
挿入　29
表内の数の指定　35
表内の合計の判別　23
列見出し
記憶域の割り当て　19, 35
検索　27
削除　24, 27, 36
名前の割り当て　36
列見出し *(続き)*
列番号の戻し　19

D

dtw_ インターフェース関数　7
dtw_ 構造体　4
dtw_ ユーティリティー　13
dtw_lei_t
構造体　4
フィールド
関数名　4
default_error_messages　5
exec_statement　5
flags　4
le_opaque_data　5
parm_data_array　5
row　6
DTW_LE_CONTINUE　9
dtw_parm_data_t
構造体　6
フィールド
parm_descriptor　6
parm_name　7
parm_value　7

E

ENVIRONMENT ステートメント
新しい言語環境用　10
構文　10
例　12

F

FUNCTION ブロック
ステートメントの実行　8
名前　4

P

parm_data_array 構造体名の割り当て　5

Printed in Japan

SB88-7410-00

日本アイ･ビー･エム株式会社
〒106-8711　東京都港区六本木3-2-12