

**Net.Data**



## **言語環境解説書**



**Net.Data**



## **言語環境解説書**

— 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、 55ページの『付録C. 特記事項』に記載する一般情報をお読みください。

原 典： VNDR-2LAN-01 (この資料番号ではオーダーできません)  
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
<b>Net.Data の言語環境について . . . . .</b>	<b>vii</b>
<b>第1章 新しい言語環境の作成 . . . . .</b>	<b>1</b>
DLL または共用ライブラリーの設計 . . . . .	1
指定すべき言語環境インターフェース . . . . .	2
入力パラメーターを処理する . . . . .	2
ユーザー要求を処理する . . . . .	3
出力パラメーターを処理する . . . . .	3
エラー条件を伝達する . . . . .	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
<b>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 . . . . .</b>	<b>13</b>
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
構成変数の管理用ユーティリティー関数 . . . . .	14
表操作のユーティリティー関数 . . . . .	14
行操作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

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
<b>付録A. Net.Data テクニカル・ライブラリー</b>	<b>39</b>
<b>付録B. 言語環境のテンプレート</b>	<b>41</b>
<b>付録C. 特記事項</b>	<b>55</b>
商標	56
用語集	57
索引	59

---

## 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 Web ページを作成するための IBM 製開発ツールです。Net.Data を使用すれば、さまざまなデータ・ソースからデータを取り込むことや、既知のプログラム言語が持つ長所を生かすことにより、動的コンテンツを含んだ 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 経由でアクセス可能なデータベースを含む関係データベース管理システムおよび非関係データベース管理システム (DBMS) の両方、ならびに Java、JavaScript、Perl、C、C++、および 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 Web ページを作成することができます。Net.Data ファミリー製品は、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・システムを実行しているマシン上で同様の機能を提供します。

Net.Data は、Web サーバー・マシン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。マクロと称する 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、これを解釈して、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予定の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 Web ページを作成します。

要求は URL (uniform resource locator) の形式で Netscape Navigator または 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、作成する関数に基づいてカスタマイズする Web ページを作成します。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 プログラム言語などで書かれているアプリケーション内の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 サーバーに渡します。Web サーバーは、そのページをネットワークを経由して表示装置のブラウザーに転送します。

Net.Data は、HTTP (HyperText Transfer Protocol) およびコモン・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構成のサーバー環境で使うことができます。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 Web サーバーと Net.Data のようなゲートウェイ・アプリケーションの起動のための業界標準インターフェースです。これらのインターフェースを使用することにより、好みのブラウザーもしくは Web サーバーを選んで Net.Data に使用することができます。

パフォーマンスの向上により、Net.Data はさまざまな Web サーバー API をサポートしています。さらに、Net.Data は Java サブレットとして立ち上げることができます。

---

## 本書について

本書では、Net.Data の言語環境インターフェース (LEI) について説明します。このインターフェースを使用すると、ユーザー独自の Net.Data 用言語環境を開発することができます。

本書では、告知後、まだ発売されてい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。(英語版)

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

## 本書の対象読者

特定企業のニーズに合わせて Net.Data の機能を拡張したいユーザーは、本書を使用して独自の Net.Data 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、以下に記載されている情報に精通していなければなりません。

- 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* 解説書に記載されている情報

## 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することが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起動することで、ユーザーのビジネス・アプリケーションに役立つ、これらの言語環境が提供する関数を使用することができます。たとえば、直接 ODBC データベースにアクセスして、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。

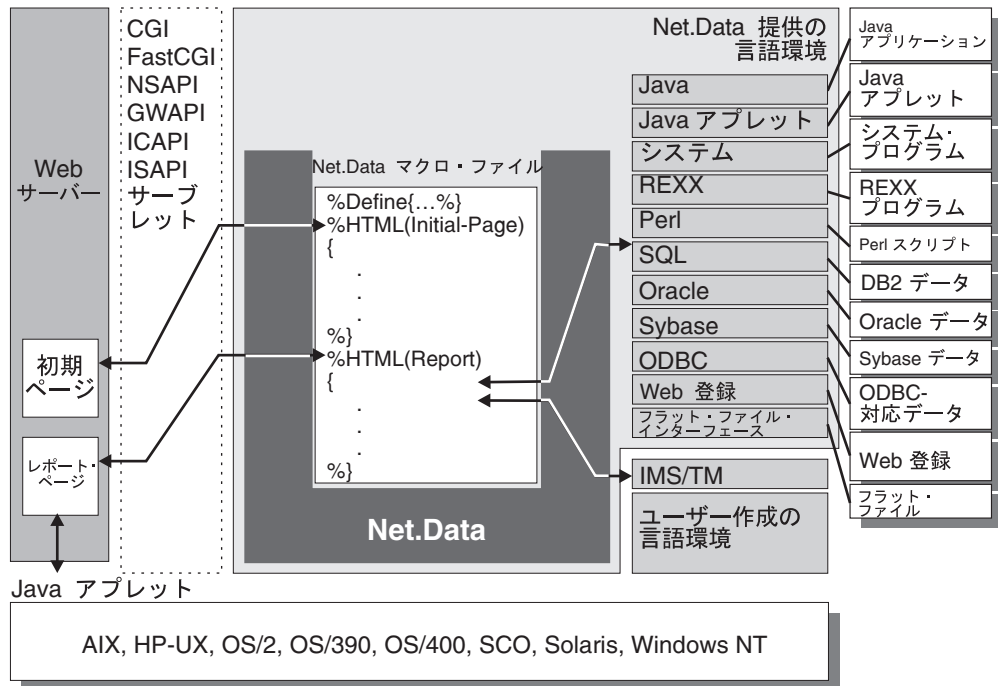


図 1. Net.Data と言語環境

本書は、新規のプログラミングおよびデータベース言語環境の作成に使用する Net.Data 言語環境インターフェースについて説明しています。 Net.Data 提供の言語環境は、 Net.Data 管理およびプログラミングの手引き (各プラットフォーム用) の言語環境の章で説明しています。

---

## 第1章 新しい言語環境の作成

Net.Data は、言語環境を、DLL ファイルもしくは共用ライブラリーとしてアクセスされる、プラグ可能なプログラミング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。Net.Data は、一連の言語環境を備えていますが、これらの言語環境がユーザーの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ユーザー独自の言語環境を作成することができます。

新しい言語環境の作成には、以下のステップが含まれます。

- 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。
- DLL または共用ライブラリーを作成するメーク・ファイルもしくは JCL を作成します。
- 提供する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 DLL または共用ライブラリーを作成します。メーク・ファイルまたは JCL を作成して DLL または共用ライブラリーを作成する方法を知るには、コンパイラーの C または C++ 関連資料を参照してください。
- すべてのインターフェースを、DLL または共用ライブラリーから外部的に使用できるようにして、Net.Data が呼び出せるようにします。
- 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。

新しい言語環境の作成を決定する前に、Net.Data 提供の言語環境がユーザーの要求を満たすかどうかを判別してください。

本章では、言語環境の設計方法を説明します。

- 『DLL または共用ライブラリーの設計』
- 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。

---

## DLL または共用ライブラリーの設計

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』で提供されているテンプレートを使用することができます。このテンプレートは、環境インターフェース関数、およびユーザーの言語環境と通信し、その言語環境との間でパラメーターをやり取りする際に Net.Data が使用する通信構造体を提供しています。

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』
- 『入力パラメーターを処理する』
- 3ページの『ユーザー要求を処理する』
- 3ページの『出力パラメーターを処理する』
- 3ページの『エラー条件を伝達する』

## 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。言語環境に行わせる内容によって選択は決まります。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ならば、言語のスクリプト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場合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か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、言語環境への最初の関数呼び出し前の各マクロ起動につき 1 回のみです。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、任意の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・システム上で稼働している場合です。

## 入力パラメーターを処理する

Net.Data 言語環境は、dtw\_execute() インターフェースを用い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勧告を参照してください。

- Net.Data の初期設定ファイルにある言語環境用 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 1 つの方法を示します。

## ユーザー要求を処理する

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。Net.Data は、関数呼び出しのたびに、dtw\_lei\_t 構造体の function\_name フィールドの言語環境に関数名を渡します。
- 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がユーザー要求を示すことができるように、指定することができます。Net.Data は、関数呼び出しのたびに、dtw\_lei\_t 構造体の parm\_data\_array フィールドの言語環境にパラメーターを渡します。
- FUNCTION ブロックの executable-statements セクションを使用。Net.Data は、関数呼び出しのたびに、dtw\_lei\_t 構造体の exec\_statement フィールドの言語環境に、FUNCTION ブロックで指定された任意の実行可能ステートメントを渡します。

## 出力パラメーターを処理する

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。

## エラー条件を伝達する

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か、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、言語環境が 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 つの構造体を用いて言語環境と通信します。言語環境はこれらの構造体処理し、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、それが呼び出す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、そして 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 構造体を指すポインターを受け取ります。dtw\_lei\_t 構造体は次の形式を持ちます。

```
typedef struct dtw_lei_t {
    char *function_name;          /* Lang. Env. Interface      */
    int  flags;                   /* Function block name      */
    char *exec_statement;        /* 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。以下の定数を用いて OR 演算を行って、flags フィールド・ポインタを指定します。

- 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を、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文字列へのポインタ
- 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

#### **parm\_data\_array**

parm\_data\_array フィールドには、dtw\_parm\_data\_t 構造体の配列へのポインタが含まれています。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、変数および関連付けられた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への変更を取り出したりするために、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列に 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です。

**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・インターフェース関数を用いて、テーブル・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 構造体を用い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、最初に明示的なパラメーター、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 構造体の形式は以下のとおりです。

```
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、次の定義を使用して 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 が動的メモリ割り当てに使用するメモリ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 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いければなりません。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、以下のステップを用い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て、したがって Webサーバーに戻る場合、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、データベースの切断や変数の解放) を行います。

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、言語環境の `dtw_initialize()` インターフェース関数をマクロごとに 1 度だけ呼び出します。最初は、その言語環境を参照する FUNCTION ブロックを呼び出します。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、それが出されます。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、言語環境によって処理され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、言語環境がテーブル・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。一度に 1 行ずつの処理であれば、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の呼び出しを指示します。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れば、パフォーマンスを最適化することができます。たとえば、ユーザーの 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され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 表の一度に 1 行処理用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 構造体で 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し示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 ユーティリティ関数の dtw\_row\_SetCols() および dtw\_row\_SetV() を使用します。

Net.Data では、`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と見なします。後続の呼び出しでは、実際のテーブル・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行します。

## 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、データベースの切断、または変数の解放などのタスクに使用します。このインターフェース関数は、任意選択です。

Net.Data 要求の処理の間、Net.Data は、Net.Data の処理が終了した場合か、エラーで Net.Data のファイルの処理が停止した場合のいずれかで、1 度言語環境の `dtw_cleanup()` インターフェース関数を呼び出します。

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` ビットを設定したことで、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で `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()` を用いてフィールドを解放します。

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。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

各言語環境は、その言語環境に特有の情報を含む Net.Data 初期設定ファイルに ENVIRONMENT ステートメントを持っています。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のリストをロードします。

Net.Data は、呼び出される際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、および資料内で使用できる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：**名前は、接頭部 DTW から始めることはできません。この接頭部は、Net.Data と一緒に出荷される言語環境に予約済みです。DTW 接頭部を使用する場合、Net.Data はユーザーの言語環境 DLL をロードすることはできません。

- *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に拡張子 *.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に拡張子 *.sl* 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に拡張子 *.so* を付けて指定します。
- OS/390、OS/2、および Windows NT では、DLL 名は拡張子 *.dll* を付けずに指定されます。
- OS/400 では、サービス・プログラム名に拡張子 *.SRVPGM* 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 と一緒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考慮してください。

- *specification*

指定を渡すパラメーターで、Net.Data が入力、出力、または入出力のパラメーターを使用するかどうかを示します。可能な値：

<b>IN</b>	入力に使用されるパラメーター
<b>OUT</b>	出力に使用されるパラメーター
<b>INOUT</b>	入出力の両方に使用されるパラメーター

- *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、パラメーターは、関数が処理を終了すると同時に変更された値を保存します。

## 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。

以下の例で、OS/2、AIX、および Windows NT の構文を使用している *Net.Data* 提供の言語環境の 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あります。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・ユーティリティ関数を説明します。

---

### 言語環境ユーティリティ関数

言語環境は、ユーティリティ関数を用いて 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関数』
- 14ページの『構成変数の管理用ユーティリティ関数』
- 14ページの『表操作のユーティリティ関数』
- 15ページの『行操作のユーティリティ関数』

### メモリー管理用のユーティリティ関数

言語環境は、メモリー管理ユーティリティ関数を用いて、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 を使用するとします。すると、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。理由は、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です。

表 1. メモリー管理ユーティリティ関数

ユーティリティ関数	説明
18ページの『 <code>dtw_malloc()</code> 』	<code>dtw_malloc()</code> 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『 <code>dtw_free()</code> 』	<code>dtw_malloc()</code> 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『 <code>dtw_strdup()</code> 』	<code>dtw_malloc()</code> 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。

## 構成変数の管理用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. テーブル・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()』	テーブル・オブジェクトを作成します。
24ページの『dtw_table_Delete()』	テーブル・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. テーブル・ユーティリティ関数 (続き)

ユーティリティ関数	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。

## 行操作のユーティリティ関数

行ユーティリティ関数は、一度に 1 行処理の間に言語環境の `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. 行ユーティリティ関数

ユーティリティ関数	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。

## ユーティリティ関数の構文参照

この節では、ユーティリティ関数、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同時に、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

<i>row</i>	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。
<i>cols</i>	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が、*src* が指し示すメモリ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

<i>row</i>	修正する行へのポインター。
<i>src</i>	設定する新しい値が入っている文字列。
<i>col</i>	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* により指定された文字列を、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

<i>table</i>	追加が行われる表へのポインター。
<i>rows</i>	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、およびテーブル・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

<i>table</i>	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

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

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

<i>table</i>	列見出しが取り出される表へのポインター。
<i>dest</i>	列見出しが入る文字列へのポインター。
<i>col</i>	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

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

<i>table</i>	修正する表へのポインター。
<i>after_col</i>	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。
<i>cols</i>	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

<i>table</i>	修正する表へのポインター。
<i>after_row</i>	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。
<i>rows</i>	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 テーブル・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

<i>table</i>	新しい表の名前。
<i>rows</i>	新しい表に割り当てる行数の初期値。
<i>cols</i>	新しい表に割り当てる列の初期数。
<i>row_lim</i>	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

<i>table</i>	照会する表へのポインター。
<i>name</i>	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()` ユーティリティ関数を使用できるのは、1 表につき 1 回のみです。その後は、`dtw_table_DeleteCol()` ユーティリティ関数か `dtw_table_InsertCol()` ユーティリティ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

<i>table</i>	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。
<i>cols</i>	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

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

<i>table</i>	値が割り当てられる表を指し示すポインター。
<i>src</i>	新しい値に割り当てられる文字ストリング。
<i>row</i>	新しい値の行番号。
<i>col</i>	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、下記の Net.Data Web サイトから入手することができます。

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

文書	説明
<i>Net.Data</i> 管理およびプログラミングの手引き OS/390	Net.Data のインストール、構成、および起動についての概念とタスク情報が入っています。また、Net.Data マクロの作成方法、Net.Data パフォーマンスの技法の使用法、Net.Data 言語環境プログラムの使用法、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・チューニングのための Net.Data ログ記録および追跡の使用法についても説明しています。
<i>Net.Data</i> 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	
<i>Net.Data</i> 管理およびプログラミングの手引き OS/400版	
<i>Net.Data</i> 解説書	
<i>Net.Data</i> 言語環境解説書	Net.Data の言語環境インターフェースについて説明しています。
<i>Net.Data</i> メッセージおよびコード	Net.Data のエラー・メッセージおよび戻りコードをリストしています。



---

## 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)

```

#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. 特記事項

以下の情報は、米国で提供される製品とサービスに関するものです。本書で説明されている製品、サービス、または機能でも、米国以外の国では提供されていないことがあります。お客様の地域で現在使用可能な製品およびサービスに関する情報については、その地域の IBM 担当員にお尋ねください。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査はお客様の責任で行っていただきます。

IBM 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 使用契約書」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:

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	+
DataJoiner	Lotus
DB2	MVS
Domino	Net.Data
IBM	OS/2
IMS	OS/390
	OS/400

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 HotJava は Sun Microsystems, Inc. の商標です。

Microsoft、Windows、Windows NT®、および Windows 95 のロゴは、Microsoft Corporation の商標です。

UNIX は、X/Open Company Ltd. にのみ許諾された米国およびその他の国の登録商標です。

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。絶対パス名は、最上位または「ルート」・ディレクトリー (スラッシュ (/) または円記号 (¥) 文字で定義されている) で始まる。

**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 をサポートし、CGI 処理の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 対応のブラウザを処理し、HTML ページが処理される際にロードされる。

**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、特定のデータもしくは、オペレーティング・システムまたはライセンス・プログラムを使用できるようになる。Net.Data は、所有権を主張できる Web サーバー API (ICAPI, GWAPI, ISAPI, および NSAPI) をサポートし、CGI 処理のパフォーマンスを向上させる。

**キャッシュ (cache).** 最近アクセスされたデータまたはディスク・スペースが入るメモリーの一部。同一データに続けてアクセスする場合の速度を上げることが目的。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)。

**CLLETTE.** Web サーバーからの要求に長期にわたって対応する、Net.Data Live コネクションの処理。この接続管理プログラムは、これらの要求に対応する CLLETTE 処理のスケジュールを行う。

**コミットメント制御 (commitment control).** Net.Data が資源の操作が作業単位の一部となっている場所で起動している処理の範囲内での、境界の確立。

**コモン・ゲートウェイ・インターフェース (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、次にブラウザがそのサーバーにアクセスするつど、送り返す情報のパケット。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、特に国籍をうたわない HTTP トランザクション間の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).** 処理のデフォルト・ディレクトリー。このディレクトリーからすべての相対パス名が解決される。

**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 server.** Lotus Corp. および IBM が提供する Web サーバーで、通常の接続および保護接続の両方を提供している。ICAPI および GWAPI は、このサーバーに付属して提供されるインターフェース。

**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。

**GWAPI.** Go Web サーバー 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。

**HTTP (hypertext transfer protocol).** Web サーバーとブラウザ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).** 世界中に開かれた TCP/IP コンピューター・ネットワーク。

**イントラネット (Intranet).** 会社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および複数の CLIETTE で構成される Net.Data の構成要素。Live コネクション は、データベースと Java 仮想マシンの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。ネットワークを通してのクライアント・アプリケーション・プログラムとサーバー間の相互作用を管理する。

**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).** オブジェクトの位置指定の方法をシステムに通知する。パス名は、ディレクトリー名の次にオブジェクト名の順序で表わされる。個々のディレクトリーおよびオブジェクト名は、スラッシュ (/) または円記号 (¥) 文字で区切られる。

**Perl.** 解釈済みプログラミング言語。

**永続性 (persistence).** 割り当てられた値をトランザクションの間中保持し続けている状態で、この場合、1 つのトランザクションが複数の Net.Data の起動にまたがっている。変数のみ、永続となることができる。加えて、コミットメント制御の影響下にある資源の操作は、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、あるいはトランザクションが完了するまで活動状態が維持される。

**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).** 文字列を保管および検索することができるリポジトリ。

**相対パス名 (relative path name).** 最上位または「ルート」・ディレクトリーで始まらないパス名。システムは、パス名が処理の現行作業ディレクトリーで始まることを前提としている。

**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であれば、1 つのトランザクションが複数の Net.Data の起動にまたがることができる。

**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。例：  
<http://www.software.ibm.com/data/net.data/index.html>

**作業単位 (unit of work).** 回復可能な操作のシーケンスで、1 つのアトミック操作として処理される。作業単位内のすべての操作は、単一の操作であるかのように完了 (コミット) または取り消し (ロールバック) することができる。コミットメント制御の影響下にある資源の操作のみ、コミットまたはロールバックすることができる。

**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

行操作のユーティリティ関数 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

インターフェース・テンプレート 41

構成 10

構造体 4

作成 1

紹介 13

初期化 8

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

ユーティリティ関数 13

構成、環境の 10

構成変数

管理用のユーティリティ関数 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

初期化、言語環境タスクの 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

動的なメモリ割り当て 7

特記事項 55

## [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

変数

受け渡し 5

解放 10

## [マ行]

メモリ管理ユーティリティ関数 13

## [ヤ行]

ユーティリティ関数

行操作 15

## ユーティリティ関数 (続き)

- 言語環境 15
- 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 56

## [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
- 列番号の戻し 33

## D

dtw\_ インターフェース関数 7

dtw\_ 構造体 4

dtw\_ ユーティリティ関数 13

dtw\_lei\_t

- 構造体 4
- フィールド
  - 関数名 4
  - default\_error\_messages 5

## dtw\_lei\_t (続き)

### フィールド (続き)

- exec\_statement 4
- 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

EXEC ステートメント、dtw\_lei\_t フラグ 5

## F

FUNCTION ブロック

- ステートメントの実行 8

- name 4

## P

parm\_data\_array 構造体名の割り当て 5

## R

rows

- 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





Printed in Japan

SB88-7368-01



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