



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、 83ページの『付録B. 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 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 1998.6

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 1998

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部 Net.Data 提供の言語環境	1
第1章 Net.Data 提供の言語環境の概要	3
第2章 Net.Data 提供の言語環境の使用	5
フラット・ファイル・インターフェースの言語環境	5
機密保護に関する考慮事項	5
FFI 組み込み関数	6
IMS Web 言語環境	8
Java アプレット言語環境	9
Java アプレットの作成	10
アプレット・タグの生成	10
Java アプレットの例	13
Net.Data Java アプレット・インターフェースの使用	14
Java アプリケーション言語環境	15
Java 言語環境のファイル構造	16
Java 関数の作成	17
Java 言語環境 CLIETTE の定義	17
Java 言語環境用 Net.Data の構成	17
マクロ・ファイルの作成と実行	18
ODBC 言語環境	19
Oracle 言語環境	20
Perl 言語環境	22
REXX 言語環境	23
SQL 言語環境	24
Sybase 言語環境	24
システム言語環境	26
Web 登録言語環境	27
Net.Data 言語環境の構成	29
第2部 非 IBM 言語環境	31
第3章 新しい言語環境の作成	33
DLL または共用ライブラリーの設計	33
指定すべき言語環境インターフェース	34
入力パラメーターを処理する	34
ユーザー要求を処理する	35
出力パラメーターを処理する	35
エラー条件を伝達する	35
言語環境通信構造体	36
dtw_lei 構造体	36

dtw_parm_data 構造体	38
言語環境インターフェース関数	39
dtw_initialize()	40
dtw_execute()	40
dtw_getNextRow()	41
dtw_cleanup()	42
言語環境ステートメントの設計	42
ENVIRONMENT ステートメントの構文	43
ENVIRONMENT ステートメントの例	43
第4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	45
言語環境ユーティリティー関数	45
メモリー管理用のユーティリティー関数	45
構成変数の管理用ユーティリティー関数	46
表操作のユーティリティー関数	46
行操作のユーティリティー関数	47
ユーティリティー関数の構文参照	47
dtw_free()	48
dtw_getvar()	49
dtw_malloc()	50
dtw_row_SetCols()	51
dtw_row_SetV()	52
dtw_strdup()	53
dtw_table_AppendRow()	54
dtw_table_Cols()	55
dtw_table_Delete()	56
dtw_table_DeleteCol()	57
dtw_table_DeleteRow()	58
dtw_table_GetN()	59
dtw_table_GetV()	60
dtw_table_InsertCol()	61
dtw_table_InsertRow()	62
dtw_table_MaxRows()	63
dtw_table_New()	64
dtw_table_QueryColnoNj()	65
dtw_table_Rows()	66
dtw_table_SetCols()	67
dtw_table_SetN()	68
dtw_table_SetV()	69
付録A. 言語環境のテンプレート	71
付録B. 特記事項	83
商標	84
用語集	85
索引	87

まえがき

Net.Data バージョン 2 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んだ Web ページを迅速に開発することができます。

Net.Data バージョン 2 は、ユーザーのインターネット・ビジネス・ソリューションの開発および活用に関与する新規機能を提供するとともに、大幅なパフォーマンスの向上を実現しています。

Net.Data について

IBM の Net.Data プロダクトを用いると、DB2、IMS、および ODBC 対応の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 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と考えることができます。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、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 または Internet Explorer などのブラウザから 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。

- ビジネス・ロジックを Perl スクリプト、C および C++ アプリケーション、もしくは REXX プログラム内でカプセル化する
- DB2 などのデータベースにアクセスする

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) などの業界標準のインターフェースをサポートしています。HTTP はブラウザと Web サーバー間で使用され、CGI は Web サーバーと Net.Data 間で使用されます。この結果、好みのブラウザもしくは Web サーバーを選んで Net.Data に使用することができます。Net.Data は、複数のオペレーティング・システム上の FastCGI および主要な Web サーバー API もサポートしています。

本書について

本書では、プログラムや関数、または DB2、Oracle、Sybase などのデータ・ソースを Net.Data マクロ・ファイルから呼び出すときに使用する Net.Data の言語環境について説明します。説明は、Net.Data で提供される各言語環境について行うとともに、ユーザー自身の言語環境の設計および作成に使用することのできる言語環境インターフェースについても説明します。

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。

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

本書の対象読者

Net.Data マクロを作成するユーザーは、本書で説明する情報を使用して、Net.Data が提供する言語環境の機能について知ることができます。本書には、Net.Data の独自の言語環境を作成するユーザーのための情報も記載されています。

本書で説明する概念を理解するには、C プログラミング言語、ならびに *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に精通していなければなりません。

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。関数呼び出しおよび SQL ステートメントを使っ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やユーティリティーにアクセス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 スクリプトを呼び出したり、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。

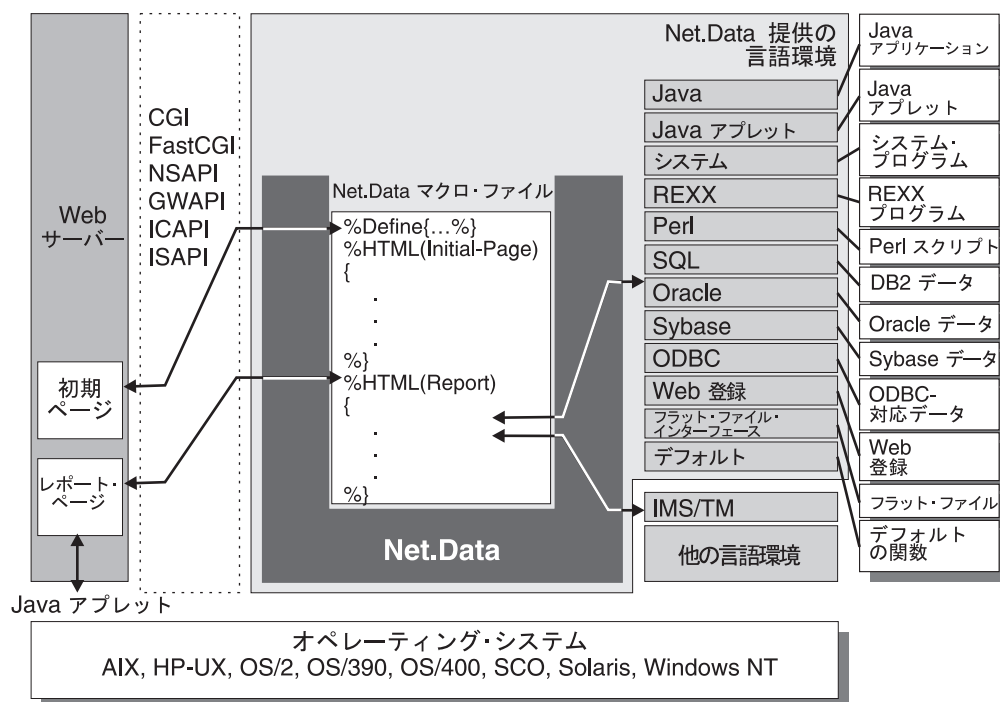


図 1. Net.Data と言語環境

Net.Data アプリケーションにおける言語環境での作業には、2 種類あります。

- Net.Data 提供の言語環境を用いての Net.Data アプリケーションの開発。
- 他ユーザーが Net.Data アプリケーションを開発するために使用するための、新しい言語またはデータベース環境の開発。

本書では、どちらの作業についてもユーザーの補助となるように編集されています。

- 1ページの『第1部 Net.Data 提供の言語環境』は、ユーザーが Net.Data アプリケーションで使用するのことができる Net.Data 提供の言語環境について説明しています。
- 31ページの『第2部 非 IBM 言語環境』は、新しい言語およびデータベース環境の作成方法について説明しています。

第1部 Net.Data 提供の言語環境

Net.Data は、データ・ソースとの情報の受け渡しを行う、いくつかの言語環境を提供します。たとえば、SQL 用の言語環境では、固有の SQL 照会をデータベースに渡します。同様に、REXX 言語環境は REXX プログラムを起動します。

この部では、各言語環境についての説明と、Net.Data 初期設定ファイルでの言語環境の構成方法についての説明を記載しています。

3ページの『第1章 Net.Data 提供の言語環境の概要』

5ページの『フラット・ファイル・インターフェースの言語環境』

8ページの『IMS Web 言語環境』

9ページの『Java アプレット言語環境』

15ページの『Java アプリケーション言語環境』

19ページの『ODBC 言語環境』

20ページの『Oracle 言語環境』

22ページの『Perl 言語環境』

23ページの『REXX 言語環境』

24ページの『SQL 言語環境』

24ページの『Sybase 言語環境』

26ページの『システム言語環境』

27ページの『Web 登録言語環境』

29ページの『Net.Data 言語環境の構成』

第1章 Net.Data 提供の言語環境の概要

Net.Data には多数の言語環境が備わっていますが、オペレーティング・システムによってはすべての環境をサポートしない場合もあります。3ページの表1に、IBM 提供の言語環境をリストします。それぞれのオペレーティング・システムで言語環境がサポートされているかどうかを判別するには、*Net.Data* 解説書のオペレーティング・システム参照付録を参照してください。それぞれのオペレーティング・システムが使用する言語環境ステートメントの詳細については、*Net.Data* の 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。

表 1. *Net.Data* の言語環境

言語環境	環境ステートメント	説明
フラット・ファイル・インターフェース	DTW_FILE	フラット・ファイル・インターフェース (FFI) は、テキスト・ファイルをデータ・ソースとしてサポートする関数を備えています。
IMS Web	HWS_LE	IMS Web 言語環境では、IMS Web を用いてIMS トランザクションを実行依頼し、トランザクションの出力をそれぞれの Web ブラウザーで受け取ることができます。
Java アプレット	DTW_APPLET	Java アプレット言語環境によって、 <i>Net.Data</i> アプリケーション内の Java アプレットを使用することができます。アプレット・タグを生成するには、アプレット・タグの修飾子とアプレットのパラメーター・リストを用意しなければなりません。
Java アプリケーション	DTW_JAVAPPS	<i>Net.Data</i> は、既存の Java アプリケーションを Java 言語環境でサポートします。
ODBC	DTW_ODBC	ODBC 言語環境は、複数のデータベース管理システムにアクセスするための ODBC インターフェースを介して SQL ステートメントを実行します。
Oracle	DTW_ORA	Oracle 言語環境によって、Oracle データに直接アクセスすることができます。
Perl	DTW_PERL	Perl 言語環境は、 <i>Net.Data</i> マクロの FUNCTION ブロックで指定されている内部 Perl スクリプトを解釈するか、あるいは別個のファイルに格納されている外部 Perl スクリプトを実行します。
REXX	DTW_REXX	REXX 言語環境は、 <i>Net.Data</i> マクロの FUNCTION ブロックで指定されている内部 REXX プログラムを解釈するか、あるいは別個のファイルに格納されている外部 REXX プログラムを実行します。
SQL	DTW_SQL	SQL 言語環境は、DB2 を介して SQL ステートメントを実行します。SQL ステートメントの結果は、表変数で戻すことができます。
Sybase	DTW_SYB	Sybase 言語環境によって、Sybase データに直接アクセスすることができます。

表 1. *Net.Data* の言語環境 (続き)

言語環境	環境ステートメント	説明
システム	DTW_SYSTEM	システム言語環境は、FUNCTION ブロックの EXEC ステートメントで識別される外部プログラムへの呼び出しをサポートします。システム言語環境は、実行するためにプログラム名とそのパラメーターをオペレーティング・システムに渡すことによって、EXEC ステートメントを解釈します。
Web 登録	DTW_WEBREG	Web 登録言語環境は、アプリケーション関連データの永続記憶用の関数を備えています。

言語環境ごとに、初期設定ファイルに ENVIRONMENT ステートメント、およびサーバーの /lib ディレクトリーもしくは /dll ディレクトリーに共用ライブラリーまたは DLL ファイルを必要とします。詳細については、*Net.Data* 管理およびプログラミングの手引き の構成の章を参照してください。

勧告: 変更する場合はその前に、初期設定ファイルをバックアップの取っておいてください。

第2章 Net.Data 提供の言語環境の使用

以下の節で、Net.Data 提供の言語環境、ならびにそれらをセットアップし使用するためのステップについて説明します。

フラット・ファイル・インターフェースの言語環境

フラット・ファイル (すなわち、平文ファイル) をデータ・ソースとして使用することにした場合は、Web サーバー上のファイルをオープン、クローズ、読み取り、書き込み、および削除するのに、フラット・ファイル・インターフェースとその関連関数を使用します。初期設定ファイルの FFI_PATH 変数用のパスを指定しなければなりません。

ファイル言語サポートは、ブラウザーを介した Web クライアントの要求に際して Web サーバーのファイルとの間で読み取りもしくは書き込みを行う場合に、FFI 関数を使用します。FFI は、ファイルを、レコード・ファイル、Net.Data マクロの表変数の行に対応する各レコード、および Net.Data マクロの表変数のフィールド値に対応するレコード内の各値として見ます。FFI はレコードをファイルから Net.Data マクロ・テーブルの行に読み取り、行を表からレコードに書き込みます。

機密保護に関する考慮事項

FFI 関数が Net.Data 初期設定ファイルの FFI_PATH ステートメントを用いてアクセスできるファイルを指定することができます。FFI が検索するのはステートメント内にリストされたパスだけであるため、他のディレクトリー内のファイルは安全です。以下にステートメントの例を示します。

```
FFI_PATH      C:¥public;.*;E:¥WWW;E:¥guest;A:
```

FFI_PATH にリストされたパスは、最初から終わりまで検索されます。Net.Data は検出した最初のコピーを使用します。FFI_PATH が初期設定ファイル内にない場合、FFI は現行ディレクトリー内のファイルの検出を試みます。Net.Data 初期設定ファイルの出荷には、FFI_PATH は付きません。

勧告:

- フラット・ファイルの操作に使用するのに適切なディレクトリーを選択します。これらのディレクトリーへの検索を制限するには、これらのディレクトリーを FFI_PATH に加える必要があります。
- 人々にマクロでの DTWF_REMOVE または他のエクスポート操作を行わせる場合は慎重を期して、現行ファイルに存在している可能性がある .dll および .cmd の拡張子付きのファイルを除去もしくは代替されないようにします。
- システムに加えるマクロに対して相応の制御を行ってシステム上のファイルを保護するための、適切なステップを踏みます。
- 匿名の FTP ユーザーがパスに書き込める FFI_PATH にはパスを指定しないでください。指定すると、それまで許されなかったアクションを可能にする Net.Data マクロがシステムに加えられる恐れがあります。
- FFI_PATH には、Net.Data 初期設定ファイルのパスを追加しないでください。

許可ヒント: Web サーバーは、FFI 組み込み関数が使用するファイルへのアクセス権限を必ず持つようにします。詳細については、*Net.Data* 管理およびプログラミングの手引き の構成の章の、*Net.Data* ファイルに対する Web サーバーのアクセス権限の指定に関する節を参照してください。

FFI 組み込み関数

この節では、FFI 組み込み関数を使用する際の使用上のヒントおよび考慮すべき問題について説明します。

一般的な考慮事項

- 平文ファイルならばいずれもインポートすることができますが、*Net.Data* マクロ構文は *Net.Data* によって解釈され、ブラウザによるテキストの形式設定には、テキスト内に存在する場合がある HTML タグが使用されます。
- FFI パラメーターは、オペレーティング・システムが大文字小文字を区別する場合に限り大文字小文字を区別します。

現行ディレクトリー

- *Net.Data* の現行ディレクトリーは、Web サーバーの構成によって決まります。CGI を使用する場合、現行ディレクトリーは *Net.Data* の実行が始まるディレクトリーで、通常は `¥www¥cgi-bin` です。Web サーバー API を使用する場合、現行ディレクトリーは変わる可能性があります。現行ディレクトリーに書き込む場合、*Net.Data* (あるいは *Net.Data* を実行しているスレッドまたは処理に関連付けられたユーザー ID) は書き込み許可を持っていないければなりません。サーバーのデフォルト要求の経路指定または資源のマッピングが変更されると、現行ディレクトリーも変更される場合があります。
- 特に Web サーバー API を使用する場合に、現行ディレクトリーの指定でお勧めする方法は、`FFI_PATH` ステートメントおよび `FILENAME` パラメーターに絶対パスを使用することです。パスにリストされたすべてのディレクトリーおよびサブディレクトリーは、初期設定ファイルの `FFI_PATH` ステートメントで定義されなければなりません。定義されなければ、*Net.Data* はそのファイルを検出することはできません。たとえば、以下のパスでは、`FFI_PATH` にディレクトリーおよびサブディレクトリー `/u/user/mydir/` がリストされている必要があります。

```
filename="/u/user/mydir/myfile.txt"
```

以下の例のようにファイル名のみを指定する場合、

```
filename="myfile.txt"
```

Net.Data は、`FFI_PATH` のパスにあるすべてのディレクトリー名とファイルの検索を最初に連結します。ファイルが見つからない場合、*Net.Data* は現行ディレクトリーにファイルがあることを前提とします。このファイルが現行ディレクトリー内にはない場合は、指定したファイル名のファイルが現行ディレクトリー内に作成されます。*Net.Data* に現行ディレクトリーへの書き込み許可がない場合は、エラーが発生します。以下の構文は使用しないでください。

```
filename="/myfile.txt"
```

DELIMITER パラメーター

- 区切り文字は、要求された変換に従ってファイルをパーツ（行の列など）に分割する際に FFI が使用する、フラグもしくは区切り記号です。
- 読み取り操作の場合、区切り文字はファイルの内容を表の行と列に分離します。書き込み操作の場合、区切り文字は表の行と列の値の終わりを示します。
Net.Data は、区切り文字を Net.Data マクロ・ストリングとして FFI に渡し、DELIMITER パラメーターに明示的にリストされない限り、文字の終わりにヌル文字を組み込みません。区切り文字にヌル文字を使用するには、2 つの二重引用符『""] を用いた空ストリングではなく、二重引用符で囲んだ斜線とゼロ『/0』を DELIMITER パラメーターに指定します。ASCITEXT の変換を指定すると、Net.Data は区切り文字として改行文字を使用し、要求された区切り文字は無視します。
- 書き込み操作に読み取り操作の場合とは異なる区切り文字を使用すると、ファイルに好ましくない変更が起こる場合があります。書き込み操作に読み取り操作で使った場合と異なる区切り文字を使用すると、Net.Data はそのファイルを新しい区切り文字で書き込みます。
- 区切り文字の最大長は、256 文字です。

FFI_PATH

- FFI_PATH のパスに含まれる印刷可能文字は、有効でなければなりません。FFI の場合、パスに疑問符 (?) または二重引用符 ("") を入れることはできません。
- FILENAME にリストされたパスのサブディレクトリーは、FFI_PATH で明示的に指定されない限り、検索されません。マクロ・ファイルのファイル名パラメーターで使用する FFI_PATH のすべてのディレクトリーおよびサブディレクトリーを指定します。以下の例で、お勧めするパス・ステートメントを示します。

例 1: すべてのディレクトリーおよびサブディレクトリーをリストする絶対パスを指定します。

```
filename="/u/usr/mydir/myfile.txt"
```

Net.Data は、FFI_PATH 内の許可されたパスを検索します。FILENAME パラメーターに割り当てられた絶対パスが誤りか利用不能の場合、Net.Data は現行ディレクトリーを検索し、ファイル名が見つからない場合は警告を出します。

例 2: 現行ディレクトリーのファイル名を指定します。

```
filename="myfile".txt
```

Net.Data は現行ディレクトリーに新しいファイルを作成します。Net.Data にファイルをディレクトリーに作成する許可がない場合、Net.Data (あるいは Net.Data を実行しているスレッドまたは処理に関連付けられたユーザー ID) は警告を出します。

DTWF_SEARCH 関数

- DTWF_SEARCH について戻される表には、3 つの列があります。最初の 2 つの列には、一致が検出された行番号と列番号が入り、最後の列には、SearchFor パラメーターで指定された文字を含む列の値が入ります。たとえば、ファイルの 4 番目の行に列 3 の一致文字が入っている場合、戻される表の行の最初の列の番号はその元のファイルの行を示す 4 となります。2 番目の列の番号は 3 となり、一致文字が含まれるファイルの列を示します。3 番目の列には列全体の値が入ります。

- *SearchFor* パラメーターには、区切り文字パラメーターの内容を組み込むことはできません。

STARTROW および ROWS パラメーター

- 関数 DTWF_DELETE、DTWF_INSERT、DTWF_UPDATE、および DTWF_WRITE の場合、*StartRow* に指定された値が最後の行より大きいと、*StartRow* は変更されて最後の行を示し、エラーが戻されます。
- 関数 DTWF_READ および DTWF_SEARCH の場合は、表の行数として *Rows* 値が戻されます。

TABLE パラメーター

- FFI 表の行の最大長は、16383 文字です。この制限の中には、Net.Data マクロ・テーブルの列ごとのヌル文字が含まれます。

TRANSFORM パラメーター

- このパラメーターは、Net.Data マクロ・テーブルの行と列の関係からファイルをパーツに分割する方法を示します。たとえば、ASCITEXT 変換は、ファイルの各行は Net.Data マクロ・テーブルの行に対応し、Net.Data マクロ・テーブルには 1 列しか含まれないことを意味します。DELIMITED 変換は、行の文字を検査して DELIMITER を検出し、DELIMITER の後は次の列の内容となることを意味します。
- ASCITEXT および DELIMITED 変換の場合、ファイルの改行文字は Net.Data マクロ・テーブルの行の終わりを示します。

ファイル・ロック

- ファイルは、DTWF_OPEN でオープンしない限りロックされません。ファイルは、ロックされないと、その読み取りと更新の間に変更される可能性があります。その結果それまでの変更が失われる恐れがあります。DTWF_OPEN を使用すると、ファイル・システムのロッキング機構を用いたマクロの実行中にファイルをオープンします。

DTWF_APPEND

- ファイルの現行の内容は、DTWF_APPEND を使用した結果、特に最後の行の最後の列の内容に影響します。ファイルの最後の行の最後の列値の後に改行が続くと、追加されるデータは、新しい行に入ります。そうでない場合は、追加されるデータは、ファイルの最後の行の一部となります。

IMS Web 言語環境

IMS Web 言語環境は、WWW 環境で IMS トランザクションを実行する場合の完全な終端間ソリューションのパーツです。IMS Web 言語環境は、以下を備えています。

- Net.Data マクロ。以下を伴います。
 - トランザクション入力データの入力に使用する HTML
 - IMS Web 言語環境を呼び出す Net.Data FUNCTION ブロック
 - トランザクションの出力を表示する HTML
- IMS Web 言語環境によって呼び出されるトランザクション DLL または共用ライブラリー

制約事項: Net.Data の IMS Web 言語環境がサポートされるのは、Net.Data が CGI アプリケーションとして実行する場合に限られます。ICAPI の Net.Data にはサポートされません。

IMS Web スタジオ・ツールは、DLL およびマクロ、ならびに DLL または共用ライブラリーを作成するための MAK ファイル用のコードを、トランザクション用の (メッセージ形式サービス) MFS ソースから生成します。DLL の実行可能形式が作成されると、このツールは Net.Data を実行している Web サーバーに DLL とこのマクロ・ファイルを移動します。これで、トランザクションは Web 環境で実行可能となりました。

IMS Web 言語環境を使用するには、次のようにします。

1. Net.Data を実行する Web サーバーに、IMS Web 実行時構成要素をインストールします。IMS Web 実行時構成要素については、*IMS Web User's Guide* を参照してください。

<http://www.software.ibm.com/data/ims/about/imsweb/document/index.html>

2. トランザクション DLL ファイルを作成します。
 - a. IMS Web スタジオ・ツールによって、トランザクションの C++、MAK、および Net.Data マクロ・ファイルを生成します。
 - b. 生成された MAK ファイルを用いて、トランザクション DLL の実行可能形式を作成します。
3. トランザクション DLL ファイル (DTWproj.dll) および Net.Data マクロ・ファイル (DTWproj.mac) を Web サーバーにコピーします。
 - a. マクロを、Net.Data がマクロの検索を行うディレクトリーに入れます。(Net.Data 管理およびプログラミングの手引き の構成の章の MACRO_PATH ステートメントを参照してください。)
 - b. トランザクション DLL を、Web サーバーが DLL または共用ライブラリーの検索を行うディレクトリーに入れます。
4. IMS Web スタジオ・ツールが生成したサンプル・ファイルのリンク、DTWproj.htm を用いて、Web サーバーの HTML ツリーの HTML ファイルを変更します。これで、このリンクを用いて Net.Data を呼び出し、Web ブラウザー上にトランザクションの入力 HTML の書式を表示することができます。トランザクション入力に記入し、書式上の SUBMIT 押しボタンを選んでトランザクションを実行し、その出力を Web ブラウザーで受け取ります。

IMS Web は、IMS TCP/IP Open Transaction Manager Access (OTMA) Connection を用いて、Web サーバーおよび IMS 環境間で通信します。詳細については、以下の IMS Web ホーム・ページを参照してください。

<http://www.software.ibm.com/data/ims/about/imsweb>

Java アプレット言語環境

Java アプレット言語環境によって、Net.Data アプリケーション内の標準 Java アプレットを使用することができます。Java アプレット言語環境を呼び出すときは、アプレットの名前を指定して、アプレットが必要とするパラメーターを渡します。言語環境は、マクロを処理し、HTML アプレット・タグを生成してから、そのアプレットを Web ブラウザーで実行します。

加えて、Net.Data には、テーブル・パラメーターを渡すときに使用する一連のインターフェースがあります。これらのインターフェースは、Java アプレット・コードから呼び出すことができ、ファイル DTW_Applet.class に入っています。

以下の節では、Java アプレット言語環境を用いて Java アプレットを実行する方法について説明します。

Java アプレットの作成

Java アプレット言語環境を使用するには、その前に、使用する予定のアプレットを決定し、必要ならそれを作成しておく必要があります。詳細については、Java 関連の資料を参照してください。Java アプレットを作成したら、それをWeb サーバー上のアプリケーション用の HTML ファイルを保持するディレクトリーに格納します。

アプレット・タグの生成

アプレット・タグを生成するマクロ・ファイルを作成するには、次のようにします。

1. マクロ・ファイルの DEFINE セクションのアプレットが必要とするパラメーターを定義します。これらのパラメーターには、アプレットの入力データとして必要とする、任意のアプレット・タグ修飾子、Net.Data 変数、および Net.Data 表または列パラメーターが組み込まれています。たとえば、以下のようになります。

```
%define{
DATABASE = "celdial"                <=Net.Data variable: the name of the database
DTW_SAVE_TABLE_IN = "MyTable"      <=The name of a valid SQL table
MyGraph.codebase = "MyMachine.ibm.com" <=Required applet parameter
MyGraph.height = "200"              <=Required applet parameter
MyGraph.width = "400"               <=Required applet parameter
MyTitle = "This is my Title"        <=Net.Data variable: the name of the Web page
%}
```

2. 任意選択: データベースへの照会を指定して、アプレットの入力データとしての結果セットを生成します。これは、図表もしくは表を生成するアプレットを使用する場合に役立ちます。たとえば、以下のようになります。

```
%SQL(A){
select age, count(age)
as ages from guests
where age >= 35
group by age
%}
```

3. Net.Data マクロの関数呼び出しを指定して、Java アプレット言語環境を呼び出し、アプレットを呼び出します。関数呼び出しは、アプレットの名前と、言語環境に渡したいパラメーターを指定します。これらのパラメーターには、アプレットの入力データとして必要とする、任意のアプレット・タグ修飾子、Net.Data 変数、および Net.Data 表または列パラメーターが組み込まれています。

以下の関数呼び出し用の構文を使用します。

```
@DTWA_AppletName(parameter1, parameter2, ..., parameterN)
```

ここで、DTWA_ 接頭部はアプレット・タグを生成する関数呼び出しを示します。AppletName は、タグを生成するアプレットの名前です。この関数呼び出しは、一般には HTML セクションにあります。たとえば、以下のようになります。

%HTML_REPORT{	<=Function calls in the HTML block
%EXEC_SQL(A)	<=EXEC statement to run the SQL query
@DTWA_MyGraph(MyTitle, DTW_COLUMN(ages) MyTable)	<=Applet function call
%}	

4. Net.Data を呼び出し、マクロ・ファイルを実行します。 *Net.Data* 管理およびプログラミングの手引き を参照して、Net.Data の呼び出し方を学んでください。

アプレット・タグ修飾子

HTML タグは常に変更されるので、Net.Data では柔軟性に富み、かつ強力なツールを使用して値を設定します。アプレット・タグ修飾子は、Net.Data マクロ・ファイル内の任意の位置で指定することができます。Net.Data は、*AppletName.qualifier* の形式を持つすべての変数を、修飾子としてのアプレット・タグに置換します。アプレット修飾子の定義例を以下に示します。

```
%define AppletName.Qualifier = "value"
```

DEFINE セクションで実際の割り当てを行う必要はありません。DTW_assign 関数で値を設定するか、SQL セクションで自動的に設定することができます。AppletName.code 変数に変数を定義しない場合は、Net.Data がデフォルトのコード・パラメーターをアプレット・タグに追加します。コード・パラメーターの値は AppletName.class です。ここで、AppletName はアプレットの名前です。

アプレット・タグ・パラメーター

関数呼び出しの Java アプレット言語環境に渡すパラメーターのリストを定義します。以下のパラメーターを渡すことができます。

- Net.Data 変数パラメーター (LIST 変数を含む)
- Net.Data テーブル・パラメーター
- 表列パラメーター

パラメーターを渡すと、Net.Data はパラメーターに割り当てた名前と値で、HTML 出力に Java アプレット PARAM タグを作成します。

Net.Data 変数パラメーター:

Net.Data 変数はパラメーターとして使用することができます。マクロの DEFINE ブロックに変数を定義し、DTWA_AppletName 関数呼び出しでその変数値を渡すと、Net.Data はその変数と同じ名前と値を持つ PARAM タグを生成します。たとえば、以下のマクロ・ステートメントであるとしします。

```
%define{
...

MyTitle = "This is my Title"
%}

%HTML_REPORT{
@DTWA_MyGraph( MyTitle, ...)
%}
```

Net.Data は以下のアプレット PARAM タグを作成します。

```
<applet code = 'MyGraph.class'
codebase = 'MyMachine.ibm.com' width = '400' height = '200' >
<param name = 'MyTitle' value = "This is my Title" >
```

...

</applet>

アプレット・テーブル・パラメーター:

Net.Data は、Java アプレット言語環境の呼び出しのつど DTW_NUMBER_OF_TABLES という名前の PARAM タグを自動的に生成して、関数呼び出しが表変数を渡したかどうかを指定します。この値は、Net.Data が関数で使用する表変数の数です。関数呼び出しで表変数が指定されない場合は、以下のタグが生成されます。

```
<param name = "DTW_NUMBER_OF_TABLES" value = "0" >
```

1 つまたは複数の Net.Data 表変数をパラメーターとして渡すことができます。DTWA_AppletName 関数呼び出しに Net.Data 表変数を指定すると、Net.Data は以下の PARAM タグを生成します。

テーブル・パラメーター・タグ:

テーブル・パラメーター・タグは以下の構文を持ちます。

```
<param name = 'DTW_TABLE_i_NAME' value = "tname" >
```

ここで、*i* は表の番号、*tname* は表の名前です。

行および列指定パラメーター・タグ:

PARAM タグが指定され、以下の構文で表の行および列の数を指定します。

```
<param name = 'DTW_tname_NUMBER_OF_ROWS' value = "rows" >  
<param name = 'DTW_tname_NUMBER_OF_COLUMNS' value = "cols" >
```

ここで、表の名前は *tname*、*rows* は表の行数、そして *cols* は表の列数です。このタグの組が、関数呼び出しで指定された固有の表ごとに生成されます。

列値パラメーター・タグ:

以下の構文では、表の列ごとに PARAM タグが生成されます。

```
<param name = 'DTW_tname_COLUMN_NAME_j' value = "cname" >
```

ここで、表名は *tname*、*j* は列番号、そして *cname* は表の列の名前です。

行値パラメーター・タグ:

以下の構文で、各行の値を含む PARAM タグが生成されます。

```
<param name = 'DTW_tname_cname_VALUE_k' value = "val" >
```

ここで、表名は *tname*、*cname* は列名、*k* は行番号、そして *val* は対応する行と列内の値と一致する値です。

表列パラメーター:

関数呼び出しで表列をパラメーターとして渡し、特定列のタグを生成することができます。Net.Data が対応するアプレット・タグを生成するのは、指定された列の場合のみです。表列パラメーターは以下の構文を使用します。

```
@DTWA_AppletName(DTW_COLUMN( x )Table)
```

ここで、*x* は表内の列名または列番号です。

表列パラメーターは、テーブル・パラメーターに定義された同じアプレット・タグを使用します。

制約事項: *DTW_tname_NUMBER_OF_COLUMNS* パラメーターの値は 1 でなければなりません。

アプレット・タグの代替テキスト

DTW_APPLET_ALTTEXT 変数を使用して、最後の param タグと APPLET タグの間に HTML を挿入することができます。APPLET タグをサポートしていないブラウザでは、この変数を含む HTML テキストが表示されます。たとえば、以下の変数定義は、

```
DTW_APPLET_ALTTEXT = "<P>Sorry, your browser is not Java-enabled."
```

以下の HTML タグおよびテキストを作成します。

```
<P>Sorry, your browser is not Java-enabled.<BR>
```

Java アプレットの例

以下の例は、Java アプレット言語環境を呼び出す Net.Data マクロ・ファイルと、言語環境が生成する結果のアプレット・タグの例証です。

Net.Data マクロ・ファイルには、Java アプレット言語環境に対する以下の関数呼び出しが入っています。

```
%define{
DATABASE = "celdial"
DTW_APPLET_ALTTEXT = "<P>Sorry, your browser is not Java-enabled."
DTW_HTML_TABLE = "yes"
DTW_SAVE_TABLE_IN = "MyTable"
MyGraph.codebase = "'MyMachine.ibm.com'"
MyGraph.height = "200"
MyGraph.width = "400"
MyTitle = "This is my Title"
%}
%SQL(A){
select age, count(age)
as ages from guests
where age >= 35
group by age
%}
%HTML_REPORT{
%EXEC_SQL(A)
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
%}
```

DEFINE セクションの Net.Data マクロの行は、アプレット・タグの最初の行を指定します。

```
MyGraph.codebase = "'MyMachine.ibm.com'"
MyGraph.height = "200"
MyGraph.width = "400"
```

言語環境は、以下の修飾子でアプレット・タグを生成します。

```
<applet code = 'MyGraph.class'
codebase = 'MyMachine.ibm.com' width = '400'
height = '200' >
```

Net.Data は、結果表、MyTable 内の Net.Data マクロ・ファイルの SQL セクションから、SQL 照会結果を戻します。この表は、DEFINE セクションで指定されます。

```
DTW_SAVE_TABLE_IN = MyTable
```

マクロのアプレットへの呼び出しは、DEFINE セクションで指定されます。

```
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
```

関数呼び出しのパラメーターに基づき、Net.Data は、列数、戻された行数、および結果行などの、結果表に関する情報の入った完全なアプレット・タグを生成します。Net.Data は、以下の例に見られるように、結果表のセルごとに 1 つのパラメーター・タグを生成します。

```
param name = 'DTW_MyTable_ages_VALUE_1' value = "4">
```

パラメーター名、*DTW_MyTable_ages_VALUE_1* は、4 の値を持つ表、MyTable 内のテーブル・セル (行 1、列 ages) を指定します。アプレットへの関数呼び出し内のキーワード DTW_COLUMN は、以下に見られる結果表 MyTable の列 ages のみが関係していることを指定します。

```
@DTWA_MyGraph( MyTitle, DTW_COLUMN(ages) MyTable )
```

以下の出力は、この例で Net.Data が生成する完全なアプレット・タグを示します。

```
<applet code = 'MyGraph.class'
codebase = 'MyMachine.ibm.com' width = '400' height = '200' >
<param name = 'MyTitle' value = "This is my Title" >
<param name = 'DTW_NUMBER_OF_TABLES' value = "1" >
<param name = 'DTW_TABLE_1_NAME' value = "MyTable" >
<param name = 'DTW_MyTable_NUMBER_OF_ROWS' value = "5" >
<param name = 'DTW_MyTable_NUMBER_OF_COLUMNS' value = "1" >
<param name = 'DTW_MyTable_COLUMN_NAME_1' value = "ages" >
<param name = 'DTW_MyTable_ages_VALUE_1' value = "4">
<param name = 'DTW_MyTable_ages_VALUE_2' value = "1">
<param name = 'DTW_MyTable_ages_VALUE_3' value = "1" >
<param name = 'DTW_MyTable_ages_VALUE_4' value = "1" >
<param name = 'DTW_MyTable_ages_VALUE_5' value = "1" >
<P>Sorry, your browser is not Java-enabled.<BR>
</applet>
```

Net.Data Java アプレット・インターフェースの使用

Net.Data には、Java アプレットとともに使用して、Java アプレット言語環境の使用時のアプレット・タグの処理で役立つ、一連のインターフェースがあります。これらのインターフェースは、Java アプレット・コードから呼び出すことができます。このインターフェースは、DTW_Applet.class で定義されます。

以下のようなインターフェースが Net.Data には備わっています。

- **int GetNumberOfTables()**。アプレット・タグで見つかった表の数を戻します。
- **String [] GetTableNames()**。アプレット・タグで見つかった表名のリストを戻します。
- **int GetNumberOfColumns(String table_name)**。表内に存在する列の数を戻します。
- **int GetNumberOfRows(String table_name)**。表内に存在する行の数を戻します。
- **String[] GetColumnNames(String table_name)**。表内の複数の列名を戻します。

- **String[][] GetTable(String table_name)**。表の行値および列値を含む 2 次元の文字列配列を返します。

インターフェースにアクセスするには、以下の例に見られるように、アプレット・コードの EXTENDS キーワードを用いて、アプレットを DTW_APPLET クラスからサブクラス化します。

```
import java.io.*;
import java.applet.Applet;

public class myDriver extends DTW_Applet
{
    public void init()
    {
        super.init();

        if (GetNumberOfTables() > 0)
        {
            String [] tables = GetTableNames();
            printTables(tables);
        }
    }

    private void printTables(String[] tables)
    {
        String table_name;

        for (int i = 0; i < tables.length; i++)
        {
            table_name = tables[i];
            printTable(table_name);
        }
    }

    private void printTable(String table_name)
    {
        int nrows = GetNumberOfRows(table_name);
        int ncols = GetNumberOfColumns(table_name);

        System.out.println("Table: " + table_name + " has " + ncols + " columns and " +
            nrows + " rows.");

        String [] col_names = GetColumnNames(table_name);

        System.out.println("-----");
        for (int i = 0; i < ncols; i++)
            System.out.print(" " + col_names[i] + " ");
        System.out.println("\n-----");

        String [][] mytable = GetTable(table_name);

        for (int j = 0; j < nrows; j++)
        {
            for (int i = 0; i < ncols; i++)
                System.out.print(" " + mytable[i][j] + " ");

            System.out.println("\n");
        }
    }
}
```

Java アプリケーション言語環境

Net.Data は、既存の Java アプリケーションを Java 言語環境でサポートします。Java アプレットおよび Java メソッド (もしくはアプリケーション) のサポートによって、Java Database Connectivity (JDBC**) API を介して DB2 にアクセスすることができます。

JDBC に関する詳細は、以下の Web サイトで入手することができます。

- IBM のソフトウェアには、Net.Data とともに JDBC を使用するために必要な JDK 1.1 以上が添付されています。

<http://www.software.ibm.com/data/db2/jdbc/>

- JavaSoft には、追加の JDBC ドライバー、JDBC API に関する資料、および最新版の JDBC があります。

<http://splash.javasoft.com/jdbc/>

Java 言語環境には、リモート・プロシージャール呼び出し (RPC) と同種のインターフェースがあります。Net.Data スtringをパラメーターとする Net.Data マクロ・ファイルから Java 関数呼び出しを出し、呼び出した Java 関数がStringを戻すことができます。Java 言語環境を使用する際は Net.Data Live コネクションを使用しなければなりません (Live コネクションの詳細については、*Net.Data 管理およびプログラミングの手引き* のパフォーマンスの章を参照してください)。Java 言語環境を使用するためには、以下のステップを完了する必要があります。これらのステップについては、以降で詳細を説明します。

1. Java 関数を作成する。
2. すべての Java 関数に Net.Data CLIETTE を作成する (Net.Data CLIETTE は、Java 関数が実行する Java 仮想マシンを立ち上げます)。
3. Live コネクション構成ファイルの CLIETTE ステートメントを定義する。
4. 接続管理プログラムを開始する。
5. Java 言語環境を呼び出す Net.Data マクロ・ファイルを実行する。

Java CLIETTE は、新規 Java 関数を導入するつど再作成しなければなりません。

Java 言語環境のファイル構造

Net.Data は、Net.Data のインストールの際にいくつかのディレクトリを作成します。これらのディレクトリには、Java 関数の作成、CLIETTE の定義、および Java 言語環境でのマクロの実行の際に必要なファイルが組み込まれています。

- UserFunctions.java というサンプルの Java 関数。
- makeClas というサンプル・ファイル。実行時、このファイルは Java 関数用の Net.Data CLIETTE クラスを作成します。
- Net.Data CLIETTE が Java 仮想マシンを立ち上げて、Java 関数を実行する際に使用する、launchjv というサンプル・ファイル。

16ページの表 2 で、オペレーティング・システム上のファイルのディレクトリとファイル名を説明します。

表 2. Java 関数の作成に使用するファイル

オペレーティング・システム	ファイル名	ディレクトリ
OS/2	UserFunctions.java	javaapps
	launchjv.com	connect
Windows NT	UserFunctions.java	javaclas
	makeClas.bat	javaclas
	launchjv.bat	connect
UNIX	UserFunctions.java	javaapps
	launchjv	javaapps

Java 関数の作成

Java 関数のサンプル・ファイル `UserFunctions.java` を変更するか、`myfile.java` という以下のサンプル・ファイルをモデルにして新しいファイルを作成します。

```
=====myfile.java=====
import mypackage.*                                <=contain your functions
public String myfctcall(...parameters from macro file...)
{
    return ( mypackage.mymethod(...parameters...));    <=high-level call to your functions
}

public String lowlevelcall(...parameters...)
{
    string result;
    .....code using many functions of your package...
    return(result)
}
```

Java 言語環境 CLIETTE の定義

サンプル・ファイル `makeClas.bat` を変更するか、新しい `.bat` ファイルを作成して、すべての Java 関数に `dtw_samp.class` という `Net.Data CLIETTE` クラスを生成します。以下の例で、バッチ・ファイル `CreateServer` が 3 つの Java 関数进行处理する方法を示します。

```
rem Batch file to create dtw_samp for Net.Data
java CreateServer dtw_samp.java UserFunctions.java myfile.java
javac dtw_samp.java
```

バッチ・ファイルは、`Stub.java` という `Net.Data` 提供のスタブ・ファイルとともに、以下のファイル进行处理して、`dtw_samp.class` を作成します。

- `dtw_samp.java`
- `UserFunctions.java`
- `myfile.java`

JDBC アプリケーションまたはアプレットを作成することは、データベースをアクセスするために `DB2 CLI` または `ODBC` を使用して C アプリケーションを作成することに似ています。アプリケーションとアプレットの主な違いは、アプリケーションでは、たとえば `DB2 Client Application Enabler` などの `DB2` と通信するための特別なソフトウェアが必要なことです。一方、アプレットは Java 対応の Web ブラウザーに依存し、クライアントに `DB2` コードをインストールする必要がありません。

システムは、JDBC を使用する前に構成を必要とします。これらの考慮事項については、`DB2 JDBC アプリケーションおよびアプレットのサポート Web サイト`で説明します。

<http://www.software.ibm.com/data/db2/jdbc/db2java.html>

Java 言語環境用 Net.Data の構成

Java 言語環境を使用するには、`Net.Data` を構成しなければなりません。以下のステップを用いて、その構成ステップを完了します。

1. Net.Data は直接 Java アプリケーションを開始することができないので、バッチ・ファイルを作成して Java アプリケーションを立ち上げます。Net.Data はこのファイルを使用してJava 仮想マシンを立ち上げ、Java 仮想マシンが Java 関数を実行します。バッチ・ファイルには java-classpath ステートメントが組み込まれていて、必要な Java パッケージ (標準パッケージとアプリケーション固有のパッケージ) が必ず見つかるようにしなければなりません。たとえば、バッチ・ファイル launchjv.bat には、以下の java-classpath が入っています。

```
java -classpath %CLASSPATH%;C:\DB2WWW\Javaclas dtw_samp %1 %2 %3 %4 %5 %6
```

2. Live コネクション構成ファイル dtwcm.cnf の Java 言語環境を処理する、CLIETTE を定義します。CLIETTE に、固有のポート番号とその関連バッチ・ファイル名を EXEC_NAME 構成変数で指定します。以下の例では、Java CLIETTE 名は DTW_JAVAPPS として定義され、EXEC_NAME 構成変数はバッチ・ファイル launchjv.bat の名前に設定されます。

```
CLIETTE DTW_JAVAPPS{
MIN_PROCESS=1                <= Required: this value must be 1 because
                               the JAVAPPS cliette is multi-threaded.
MAX_PROCESS=1                <= Required: this value must be 1 because
                               the JAVAPPS cliette is multi-threaded.
START_PRIVATE_PORT=5100      <= Must be a unique port number
START_PUBLIC_PORT=5300       <= Must be a unique port number
EXEC_NAME=launchjv.bat       <= The name of the batch file that includes the
                               classpath statements
}
```

Net.Data 接続管理プログラムを開始すると、Net.Data は構成ファイルで指定された Java CLIETTE を開始します。CLIETTE は、Net.Data マクロ・アプリケーションからの Java 言語環境要求の処理に使用可能になります。

3. ステートメントに各 CLIETTE 名を加えることによって、Net.Data 初期設定ファイル db2www.ini の DTW_JAVAPPS ENVIRONMENT パス・ステートメントを更新します。たとえば、以下のようにします。

```
ENVIRONMENT DTW_JAVAPPS ( OUT RETURN_CODE ) CLIETTE "DTW_JAVAPPS"
```

マクロ・ファイルの作成と実行

Java 関数の作成、CLIETTE クラスの定義、Net.Data の構成が終わると、Java 関数への参照が入ったマクロ・ファイルを実行することができます。

1. Java 関数を呼び出すマクロ・ファイルを作成します。たとえば、関数呼び出し *myfctcall* は、CLIETTE の DTW_JAVAPPS を用いて、Net.Data とともに提供されるサンプル関数を呼び出します。

```
%function (DTW_JAVAPPS) myfctcall( ...parameters from macro file ....)

%{ to call the sample provided with Net.Data %}
%function (DTW_JAVAPPS) reverse_line1(str);

%HTML_REPORT{
you should see the string "Hello World" in reverse.
@reverse_line("Hello World")
You should have the result of your function call.
@myfctcall( ... ....)
%}
```

2. 接続管理プログラムを開始します。接続管理プログラムの詳細については、*Net.Data* 管理およびプログラミングの手引き のパフォーマンスの章を参照してください。
3. アンカー参照、HTML の書式、もしくは URLステートメントを用いて、マクロを立ち上げ、*Net.Data* を呼び出します。たとえば、以下の URL ステートメントによって、*Net.Data* マクロ・ファイル *mymacro.mac* を呼び出します。

`http://myserver/cgi-bin/dt2www/mymacro.mac/report`

ODBC 言語環境

Open Database Connectivity (ODBC) 言語環境は、ODBC インターフェースを介して SQL ステートメントを実行します。ODBC は、1 つのアプリケーションが複数のデータベース管理システムにアクセスできるようにする X/Open SQL CAE 仕様に基づいています。

ODBC 言語環境を使用するには、ODBC ドライバーおよびドライバー・マネージャーがなければなりません。ODBC ドライバーの資料には、ODBC 環境をインストールおよび構成する方法が記載されています。

ODBC 環境での SQL ステートメントの送信は、他の *Net.Data* 関数と類似しています。以下の例は、ODBC データ・ソースであるデータベースに、複数の SQL ステートメントを送信する *Net.Data* マクロです。DATABASE 変数を使用するオペレーティング・システムは、ODBC.INI ファイルのデータ・ソースと同じデータベースを指定しなければなりません。

```
%DEFINE {  
    DATABASE="qesq1"  
    SHOWSQL="YES"  
    table="int_null"  
    LOGIN="netdata1"  
    PASSWORD="ibmdb2"  
%}  
  
%function(dtw_odbc) sql1() {  
create table int_null (int1 int, int2 int)  
%}  
  
%function(dtw_odbc) sql2() {  
insert into $(table) (int1) values (111)  
%}  
  
%function(DTW_odbc) sql3() {  
insert into $(table) (int2) values (222)  
%}  
  
%function(dtw_odbc) sql4() {  
select * from $(table)  
%}  
  
%function(dtw_odbc) sql5() {  
drop table $(table)  
%}  
  
%HTML(REPORT) {  
@sql1()
```

```
@sql2()  
@sql3()  
@sql4()  
%}
```

Oracle 言語環境

Oracle 言語環境には、Oracle データに対する固有のアクセス権限があります。Oracle 表には、CGI、FastCGI、NSAPI、ISAPI、または GWAPI モードで稼働する Net.Data からアクセスすることができます。

制約事項:

- ストアード・プロシージャは、この言語環境を介してはサポートされません。
- DATABASE 変数は、Oracle データベースのアクセスには使用されません。
- LOGIN 変数には、Oracle データベースのインスタンス名が入っていなければなりません。たとえば、*ora73* は以下の LOGIN 変数の定義済みインスタンス名です。

```
LOGON=admin@ora73
```

Net.Data から Oracle にアクセスするには、次のようにします。

1. Net.Data 初期設定ファイルの ENVIRONMENT ステートメントが Oracle 言語環境に対して正しいことを検証します。ステップならびに例については、*Net.Data 管理およびプログラミングの手引き* の構成の章を参照してください。
2. 以下のように、Oracle の適切な構成要素がインストールされ、かつ作動していることを確認します。

- a. SQL*Net がまだインストールされていない場合は、Net.Data がインストールされているマシンにインストールします。詳細については、以下の URL を参照してください。

```
http://www.oracle.com/products/networking/html/stdn\_sqlnet.html
```

- b. Oracle *tnsping* 関数が、Web サーバーが使用する場合と同じ機密保護許可で利用できることを検証します。検証する場合は、Web サーバーのユーザー ID と型でログオンします。

```
tnsping oracle-instance-name
```

ここで、*oracle-instance-name* は Net.Data マクロがアクセスする Oracle システムの名前です。

Web サーバーがシステム権限のもとで稼働する場合は、Windows NT の *tnsping* 関数は検証できない場合があります。その場合は、このステップをスキップしてください。

- c. Oracle 表に、Web サーバーが使用する場合と同じ機密保護許可でアクセスできることを検証します。検証する場合は、SQL*Plus 行コマンド・ツールを用いて、SQL SELECT ステートメントを入力し、Web サーバーの権限付きの SQL SELECT ステートメントで Oracle 表にアクセスします。たとえば、以下のようになります。

```
SELECT * FROM tablename
```

Web サーバーがシステム権限のもとで稼働する場合は、Windows NT のテーブル・アクセスは検証できない場合があります。その場合は、このステップをスキップしてください。

障害追及: 上記のステップが失敗したら、先へ進まないでください。いずれかのステップが失敗した場合は、Oracle の構成をチェックしてください。

3. Oracle の環境変数が Web サーバーのプロセスで正しく設定されるようにします。

- AIX の場合は、`/etc/environment` ファイルに以下の数行を書き込んでください。

```
ORACLE_SID=oracle-instance-name
ORACLE_HOME=oracle-runtime-library-directory
```

- Windows NT の場合は、「System Properties Control」パネルを用いて以下の環境変数を追加してください。

```
ORACLE_SID=oracle-instance-name
ORACLE_HOME=oracle-runtime-library-directory
```

ヒント: 各国語サポートや 2 フェーズ・コミットなど、使用を予定している Oracle 機能によっては、他の Oracle 環境変数用に数行の追加が必要になる場合があります。これらの環境変数の詳細については、Oracle の管理資料を参照してください。

4. Net.Data から Oracle への接続をテストします。Net.Data マクロ・ファイルで、LOGIN 変数および PASSWORD 変数に適切な値を指定します。Oracle データベースにアクセスするときは、DATABASE 変数を定義しないでください。以下は、マクロ・ファイル内の接続ステートメントの例です。

```
%DEFINE LOGIN=user_ID@remote-oracle-instance-name
%DEFINE PASSWORD=password
```

ローカル Oracle インスタンス:

ローカル Oracle インスタンスのみにアクセスする場合は、以下の例のように、リモート Oracle インスタンス名をログイン・ユーザー ID の一部として指定しないでください。

```
%DEFINE LOGIN=user_ID
%DEFINE PASSWORD=password
```

Live コネクション:

Live コネクションを使用する場合は、Live コネクション構成ファイルで LOGIN および PASSWORD を指定できますが、機密保護の目的からはお勧めできません。たとえば、以下のようになります。

```
LOGIN=user_ID
PASSWORD=password
```

ヒント: Oracle には DATABASE 変数を指定しないでください。

5. 以下の例のように、CGI シェル・スクリプトを実行し、Oracle インスタンスに Web サーバーからアクセスできるようにして、構成をテストします。

```
#!/bin/sh
echo "content-type; text/html"
echo
echo "< html><< pre>"
```



```
set
echo "</pre>< p>< pre>"
tnsping oracle-instance-name
echo
```

あるいは、以下の例のように、Net.Data マクロから直接 *tnsping* を実行することができます。

```
%DEFINE testora = %exec "tnsping oracle-instance-name"
%HTML (report){
< P>About to test Oracle access with tnsping.
< hr>
$(testora)
< hr>
< P>The Oracle test is complete.
%}
```

障害追及:

検査ステップが失敗した場合は、以下の項目を検証して、それまでのステップがすべて正常であったかをチェックします。

- Oracle 構成をチェックします。
- Oracle 環境変数の構文が正しく、かつ脱落変数がないことを検証します。
- Oracle 接続をチェックして、正しいユーザー ID とパスワードが入力されたことを確認します。

それでも検査ステップが失敗したら、IBM サービスに連絡してください。

例:

アクセス検査ステップが完了すると、以下の例のように、マクロ・ファイルの関数によって Oracle 言語環境を呼び出すことができます。

```
%FUNCTION(DTW_ORA) STL1() {
insert into ${tablename} (int1,int2) values (111,NULL)
%}
```

Perl 言語環境

Perl 言語環境は、Net.Data マクロの FUNCTION ブロックで指定したインライン Perl スクリプトを解釈することもできれば、サーバーの個別ファイルに格納されている外部 Perl スクリプトを処理することもできます。外部 Perl スクリプトの呼び出しは、たとえば次のように、EXEC ステートメントによって FUNCTION ブロック内で識別されます。

```
%EXEC{ perl-script-name [optional parameters] %}
```

Perl 言語環境は、Net.Data 変数を直接渡したり、検索したりすることはできないため、変数は以下のようにして Perl スクリプトで使えるようになります。

- Net.Data は、入力パラメーターを環境変数として Perl スクリプトに渡します。Perl スクリプトは、Perl 結合配列を読み取ることで、パラメーターを検索することができます。
- Perl スクリプトは、Net.Data が環境変数 DTWPIPE で名前を渡す、名前付きパイプに書き込むことによって、出力パラメーターを言語環境に戻します。データを名前付きパイプに書き込む場合は、DEFINE ステートメント構文を使用します。


```
name = value
```

データ項目が複数の場合は、各項目を改行もしくはブランク文字で分離します。

変数名の持つ名前が出力パラメーターと同じで、上記の構文を使用すると、現行値は新しい値に置き換わります。変数名が出力パラメーターと一致しない場合、Net.Data はそれを無視します。

以下の例で、Net.Data がマクロ・ファイルから変数を渡す方法を示します。

```
%FUNCTION(DTW_PERL) today() RETURNS(result) {
    $date = 'date';
    chop $date;
    open(DTW, "> $ENV{DTWPIPE}") || die "Could not open: $!";
    print DTW "result = ¥$date¥¥\n";
}%
%HTML(INPUT) {
    @today()
}%
```

Perl スクリプトが `today.pr1` という外部ファイルにある場合は、次の例のように、同じ関数を書き込むことができます。

```
%FUNCTION(DTW_PERL) today() RETURNS(result) {
    %EXEC { today.pr1 %}
}%
```

Perl 言語環境プログラムは、その Net.Data 名でテーブル・パラメーターの値にアクセスします。表 T の列見出しは T_N_i、フィールド値は T_V_i_j です。表 T の行数と列数は T_ROWS と T_COLS です。

FUNCTION セクションの場合と同様に、REPORT および MESSAGE ブロックも認められます。これらのブロックは、言語環境ではなく、Net.Data によって処理されます。しかし、Perl プログラムは、標準出力ストリームにテキストを書き込んで、出力 HTML の書式を直接に操作することができます。

許可ヒント: Web サーバーが、Perl 解釈プログラムの正しいバージョンを含む、この言語環境が参照する外部実行可能ファイルへのアクセス権限を必ず持つようにします。詳細については、*Net.Data 管理およびプログラミングの手引き* の構成の章の、Net.Data ファイルに対する Web サーバーのアクセス権限の指定に関する節を参照してください。

REXX 言語環境

REXX 言語環境は、Net.Data マクロの FUNCTION ブロックで指定されているインライン REXX プログラムを解釈したり、別個のファイルに格納されている外部 REXX プログラムを実行したりすることができます。外部 REXX プログラムの呼び出しは、以下のようなステートメントによって FUNCTION ブロック内で識別されます。

```
%EXEC{ REXX-program-file-name [optional parameters] %}
```

REXX 言語環境は、RexxStart() API を使用して REXX インタープリターに指定のファイルを実行するように伝え、その後、プログラムにファイル名とパラメーターを (コマンド行に入力されたかのように) 渡します。REXX プログラムでは、すべてのパラメーターが ARG[1] として受け取られます。

変数置換は、FUNCTION ブロックの executable-statements セクションでのみ実行されます。ただし、パラメーターは、REXX プログラムが FUNCTION ブロックで内部的に定義されているか、別個のファイルで外部的に定義されているかに関係なく、プログラムにとってアクセス可能になります。REXX 言語環境は、REXX 言語処理プログラムの RexxVariablePool() 関数を使用して、Net.Data 変数を REXX プログラムと共有します。このため、REXX プログラムは、パラメーター・リストで識別された Net.Data 変数を直接に操作することができます。

REXX プログラムは、テーブル・パラメーターの値を REXX stem 変数としてアクセスします。REXX プログラムに対して、表 T の列見出しは T_N.i、フィールド値は T_V.i.j です。表 T の行数と列数は T_ROWS と T_COLS です。

許可ヒント: Web サーバーが、この言語環境が参照する外部実行可能ファイルへのアクセス権限を必ず持つようにします。詳細については、*Net.Data 管理およびプログラミングの手引き* の構成の章の、Net.Data ファイルに対する Web サーバーのアクセス権限の指定に関する節を参照してください。

SQL 言語環境

SQL 言語環境は、DB2 を介して SQL ステートメントを実行するために使用されます。SQL ステートメントの結果は、Net.Data のデフォルト・テーブルまたはユーザーが指定した表で戻すことができます。

Net.Data は、許可したすべての SQL ステートメントをサポートします。Net.Data を CGI アプリケーションとして呼び出すときは、HTML セクションごとに 1 つのデータベースに接続することができますが、データベース名は DATABASE 変数で指定しなければなりません (OS/390 の場合を除く)。DB2 データベースが Web サーバーと同じマシンに存在する場合は、追加のセットアップは不要です。存在しない場合は、使用するオペレーティング・システムに応じて、Client Application Enabler (CAE) を用いてリモート・データベースにアクセスすることも、Database Connection Services (DDCS) を用いて DB2 がサポートするすべてのトランザクション・サポートを得ることもできます。DataJoiner を用いて他のデータベースにアクセスできる場合もあります。DataJoiner を使用すると、2 フェーズ・コミットをサポートするデータベースで 2 フェーズ・コミットを使用することができます。

Sybase 言語環境

Sybase 言語環境には、Sybase データに対する固有のアクセス権限があります。Sybase 表には、CGI、FastCGI、NSAPI、ISAPI、または GWAPI モードで稼働する Net.Data からアクセスすることができます。

制約事項:

Sybase 言語環境では、イメージやオーディオなどの大規模なオブジェクトはサポートされていません。ストアド・プロシージャは、SELECT ステートメントを使用しないプロシージャの場合にのみサポートされます。

Net.Data から Sybase にアクセスするには、次のようにします。

1. Net.Data 初期設定ファイルの ENVIRONMENT ステートメントが Sybase 言語環境に対して正しいことを検証します。ステップならびに例については、*Net.Data* 管理およびプログラミングの手引き の構成の章を参照してください。

2. 以下のように、Sybase の適切な構成要素がインストールされ、かつ作動していることを確認します。

a. Sybase Open Client がまだインストールされていない場合は、Net.Data がインストールされているマシンにインストールします。詳細については、Sybase Open Client 関連の資料を参照してください。

b. Sybase *ping* 関数が Web サーバーが使用する場合と同じ機密保護許可で利用できることを検証します。検証する場合は、Web サーバーのユーザー ID と型でログオンします。

```
ping sybase-instance-name
```

ここで、*sybase-instance-name* は Net.Data マクロがアクセスする Sybase システムの名前です。

Web サーバーがシステム権限のもとで稼働する場合は、Windows NT の *ping* 関数は検証できない場合があります。その場合は、このステップをスキップしてください。

c. Sybase 表に、Web サーバーが使用する場合と同じ機密保護許可でアクセスできることを検証します。検証する場合は、SQL 行コマンド・ツールを用いて、SQL SELECT ステートメントを入力し、Web サーバーの権限付きの Sybase 表にアクセスします。たとえば、以下のようにします。

```
SELECT * FROM tablename
```

Web サーバーがシステム権限のもとで稼働する場合は、Windows NT のテーブル・アクセスは検証できない場合があります。その場合は、このステップをスキップしてください。

障害追及: 上記のステップが失敗したら、先へ進まないでください。いずれかのステップが失敗した場合は、Sybase の構成をチェックしてください。

3. Sybase の環境変数が Web サーバーのプロセスで正しく設定されるようにします。

• AIX の場合は、*/etc/environment* ファイルに以下の数行を書き込んでください。

```
DSQUERY=sybase-instance-name  
SYBASE=sybase-runtime-library-directory
```

• Windows NT の場合は、「System Properties Control」パネルを用いて以下の環境変数を追加してください。

```
DSQUERY=sybase-instance-name  
SYBASE=sybase-runtime-library-directory
```

ヒント: 各国語サポートや 2 フェーズ・コミットなど、使用を予定している Sybase 機能によっては、他の Sybase 環境変数用に数行の追加が必要になる場合があります。これらの環境変数の詳細については、Sybase の管理資料を参照してください。

4. Net.Data から Sybase への接続をテストします。Net.Data マクロ・ファイルで、LOGIN、PASSWORD、および DATABASE 変数に適切な値を指定します。以下は、マクロ・ファイル内の接続ステートメントの例です。

```
%DEFINE DATABASE=database-name
%DEFINE LOGIN=user_ID@remote-sybase-instance-name
%DEFINE PASSWORD=password
```

Live コネクション: Live コネクションを使用する場合は、Live コネクション構成ファイルで LOGIN および PASSWORD を指定できますが、機密保護の目的からはお勧めできません。たとえば、以下のようにします。

```
DATABASE=database-name
LOGIN=user_ID
PASSWORD=password
```

5. 以下の例のように、CGI シェル・スクリプトを実行し、Sybase インスタンスに Web サーバーからアクセスできるようにして、構成をテストします。

```
#!/bin/sh
echo "content-type; text/html"
echo
echo "< html>< pre>"
set
echo "</pre>< p>< pre>"
isql -u user_ID -p password << EOFF
SELECT * FROM tablename
EOFF
echo
```

障害追及:

検査ステップが失敗した場合は、以下の項目を検証して、それまでのステップがすべて正常であったかをチェックします。

- Sybase 構成をチェックします。
- Sybase 環境変数の構文が正しく、かつ脱落変数がないことを検証します。
- Sybase 接続をチェックして、正しいユーザー ID とパスワードが入力されたことを確認します。

それでも検査ステップが失敗したら、IBM サービスに連絡してください。

例:

アクセス検査ステップが完了すると、以下の例のように、マクロ・ファイルの関数によって Sybase 言語環境を呼び出すことができます。

```
%function(DTW_SYB) STL1() {
insert into $(tablename) (int1,int2) values (111,NULL)
%}
```

システム言語環境

システム言語環境は、FUNCTION ブロックの EXEC ステートメントで識別される外部プログラムへの呼び出しをサポートする、Net.Data 定義の環境です。

システム言語環境は、C 言語の system() 関数呼び出しを使用した処理を行うために、プログラム名とパラメーターをオペレーティング・システムに渡すことによって、EXEC ステートメントを解釈します。この方式の場合、外部プログラムは、REXX 言語環境が行うように Net.Data との間で変数を直接やり取りすることができません。したがって、Net.Data は、以下の方式で変数を処理します。

- Net.Data は、入力パラメーターを環境変数として外部プログラムに渡し、外部プログラムはそれを取り出します。
 - UNIX CSHELL スクリプトは、\$x のように、環境変数の前にドル記号 (\$) を付けることによって環境変数を参照します。
 - Perl 言語スクリプトは、%ENV{'x'} のように、結合配列 ENV を参照することで環境変数を参照します。
 - DOS のバッチ・ファイルでは、%x% のように、パーセント記号 (%) で囲んで変数名を参照します。
- ほとんどのオペレーティング・システムは、Net.Data が環境変数 DTWPIPE で名前を渡す、名前付きパイプに書き込むことによって、出力パラメーターを言語環境に戻します。(Net.Data for OS/400 では、環境変数を使用してパラメーターを言語環境に戻します。) データを名前付きパイプに書き込む場合は、DEFINE ステートメント構文を使用します。

`name = value`

データ項目が複数の場合は、各項目を改行もしくはブランク文字で分離します。

変数名が出力パラメーターに一致する場合は、現行値が新しい値によって置換されます。Net.Data は、出力パラメーターに一致しない変数名は無視します。

システム言語環境は、テーブル・パラメーターの値にアクセスする際に、それらの Net.Data 名を使用します。表 T の列見出しは T_N_i、フィールド値は T_V_i_j です。表 T の行数と列数は T_ROWS と T_COLS です。

許可ヒント: Web サーバーが、システム言語環境から呼び出される外部実行可能ファイルへのアクセス権限を必ず持つようにします。詳細については、*Net.Data 管理およびプログラミングの手引き* の構成の章の、Net.Data ファイルに対する Web サーバーのアクセス権限の指定に関する節を参照してください。

Web 登録言語環境

Net.Data Web 登録は、アプリケーション関連データ用の永続的な記憶域を備えています。Web 登録は、構成情報および、Web ベースのアプリケーションが実行時に動的にアクセスできる他のデータを保管するのに使用することができます。Web 登録にアクセスできるのは、Net.Data を用いた Net.Data マクロおよび Web 登録組み込みサポートを使用した場合と、この目的で作成された CGI プログラムからの場合に限られます。Web 登録は、オペレーティング・システムのサブセットで使用できます。*Net.Data 解説書* の Net.Data オペレーティング・システムの参照付録を参照してください。

標準の Web ページ開発では、URL はそのページの HTML ソース内に直接入れられる必要があります。これで、リンクの変更は困難になります。静的な特質によっても、Web ページに簡単に入れることができるリンクの型は制限されます。Web 登録を用いてアプリケーション関連のデータ、たとえば URL を保管する方法は、動的に設定されたリンクで HTML ページを生成する際に役立ちます。

登録での情報の保管、保持は、その登録への書き込みアクセス権限を持つアプリケーション開発者や Web 管理者が行うことができます。アプリケーションは、実行時にその関連登録から情報を取り出します。これで柔軟なアプリケーションの設計が

生かされ、アプリケーションとサーバーを移動することができます。動的に設定されたリンクを用いた HTML ページの作成には、Net.Data マクロを使用することができます。

情報は、登録項目の形式で Web 登録に保管されます。各登録項目は、1 組みの文字ストリング、すなわち RegistryVariable ストリングと対応する RegistryData ストリングから構成されます。1 組みのストリングで表すことができる情報は、すべて登録項目として保管することができます。Net.Data は、変数ストリングを検索キーとして使用して、登録の特定項目を探し、検索することができます。

Web 登録のサンプル内容を、28ページの表3 に示しました。

表3. サンプル Web 登録

CompanyName	WorldConnect
Server	ftp.einet.net
JohnDoe/foreground	Green
CompanyURL/IBM Corp.	http://www.ibm.com
CompanyURL/Sun Microsystems Corp.	http://www.sun.com
CompanyURL/Digital Equipment Corp.	http://www.dec.com
JaneDoe/Home_page	http://jane.info.net

Web 登録の使用に際しては、以下のように考慮するべきいくつかの理由があります。

- Web 登録を使用すると、サーバーと URL の別名を保管して、アプリケーションおよびサーバーの再配置を容易にすることができます。
- アプリケーション開発者は、Web ベースのアプリケーションを、URL などの登録で事前定義されたデータとともに出荷することができます。エンド・ユーザーは、登録データを変更して、アプリケーションの振る舞いを変更することができます。
- Web 登録を使用すると、プロダクト名、各国語、製造業者、等々に基づいて URL 検索を行うことができます。

Web 登録の索引項目は、以下の構文を用いて、RegistryVariable ストリングに追加の索引ストリングが追加された項目です。

RegistryVariable/Index

ユーザーは、索引項目を処理するように設計された組み込み関数に、別のパラメーターの索引ストリングの値を指定します。複数の索引登録項目が、同じ RegistryVariable ストリング値を持つことができますが、異なる索引ストリングの値を持つことで一意性を保つことができます。

表4. サンプルの索引 Web 登録

Smith/Company_URL	http://www.ibm.link.ibm.com
Smith/Home_page	http://www.advantis.com

上記の2つの索引項目は同じ RegistryVariable ストリング値 Smith を持っていますが、どちらの場合も索引ストリングは異なります。これらは、Web 登録関数によって2つの異なる項目として扱われます。

Net.Data 言語環境の構成

言語環境を Net.Data に使用するには、Net.Data 初期設定ファイルを構成しておかなければならず、Live コネクションを使用する場合は、Live コネクション構成ファイルを構成しなければなりません。

以下は、それらのタスクの概要です。Net.Data 言語環境の構成方法の詳細については、*Net.Data* 管理およびプログラミングの手引き の構成の章を参照してください。さらに、特殊な構成の説明については、前の言語環境の節をチェックしてください。

- Net.Data 初期設定ファイル `db2www.ini` の ENVIRONMENT ステートメントを検証もしくは更新します。これらのステートメントについては、*Net.Data* 管理およびプログラミングの手引き の『Net.Data の構成』の章で説明しています。
- Live コネクション・ファイル `dtwcm.cnf` にデータベースの CLIETTE または Java アプリケーション言語環境を定義します。CLIETTE の定義については、*Net.Data* 管理およびプログラミングの手引き の構成の章で説明しています。

第2部 非 IBM 言語環境

言語環境の提供に加え、Net.Data ではユーザー独自の言語およびデータベースを追加することができます。Net.Data は、Net.Data 実行可能ファイルとは別のダイナミック・リンク・ライブラリーまたは共用ライブラリーとして、言語環境にアクセスします。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・セットをサポートしなければなりません。以下の章では、新しい言語環境の作成方法と、言語環境プログラミングのインターフェースおよび環境について説明します。

33ページの『第3章 新しい言語環境の作成』

42ページの『言語環境ステートメントの設計』

45ページの『第4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』

第3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、それがアプリケーションのニーズに合わない場合は、新しい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を決定する前に、Net.Data とともに出荷されている IBM 提供の言語環境がユーザーのニーズを満たすかどうかを判別してください。

新しい言語環境の作成を決定した場合は、以下のステップを行わなければなりません。

- 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・ファイルの作成方法、ならびに DLL または共用ライブラリーの作成方法を知るには、オペレーティング・システムの 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。詳細については、42ページの『言語環境ステートメントの設計』を参照してください。
- 新しい言語環境を使用する Net.Data マクロ・ファイルに関数を追加します。

本章では、言語環境の設計方法を説明します。

- 33ページの『DLL または共用ライブラリーの設計』
- 36ページの『言語環境通信構造体』
- 39ページの『言語環境インターフェース関数』
- 42ページの『言語環境ステートメントの設計』

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、71ページの『付録A. 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更新して、環境インターフェース関数、および言語環境と通信し、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組み込みます。

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、45ページの『第4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。

- 34ページの『指定すべき言語環境インターフェース』
- 34ページの『入力パラメーターを処理する』
- 35ページの『ユーザー要求を処理する』
- 35ページの『出力パラメーターを処理する』
- 35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・ファイル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 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、マクロが異常終了したときにエラー処理を行わせたい場合に、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、マクロ起動ごとに 1 回のみです。

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に限られます。

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` 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。

- 初期設定ファイル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` 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。

71ページの『付録A. 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` 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` 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` 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` 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。71ページの『付録A. 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 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。71ページの『付録A. 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
- dtw_parm_data

Net.Data は、言語環境インターフェース構造体 (たとえば、dtw_lei) 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 構造体

各言語環境のインターフェース関数は、dtw_lei 構造体を指すポインターを受け取ります。dtw_lei 構造体は次の形式を持ちます。

```
typedef struct dtw_lei {
    char *function_name; /* Lang. Env. Interface */
    int flags; /* Function block name */
    /* 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 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で、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 構造体の配列へのポインターが含まれています。配列は、ゼロの入った parm_data 構造体で終わります。dtw_parm_data 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、38ページの『dtw_parm_data 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 用の予約フィールドです。

dtw_parm_data 構造体

Net.Data は、dtw_parm_data 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
- 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
- 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 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 構造体の形式は以下のとおりです。

```
typedef struct dtw_parm_data {          /* 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 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 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を持た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・ファイル 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。

- 40ページの『dtw_initialize()』
- 40ページの『dtw_execute()』
- 41ページの『dtw_getNextRow()』
- 42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 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・ファイル 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 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、41ページの『`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、SQL 言語環境の `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します。

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` 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` 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。たとえば、データベースの切断もしくは変数の解放です。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 のマクロ・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 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 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ーで 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初期設定ファイル DB2WWW.INI 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ー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・ファイル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 ([usage parameter, ...])
```

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。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。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。

- *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。Windows NT および OS/2 の場合、DLL 名は拡張子 *.dll* を付けずに指定されます。AIX では、共用オブジェクトの名前に拡張子 *.o* を付けて指定します。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。OS/390 では、DLL ファイルには拡張子がありません。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。

- *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* 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 の場合の構文を示します。

```
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)   ( OUT RETURN_CODE ) CLIETTE "DTW_JAVAPPS"
| ENVIRONMENT (DTW_PERL)      DTWPERL    ( OUT RETURN_CODE )
| ENVIRONMENT (DTW_REXX)      DTWREXX    ( OUT RETURN_CODE )
| ENVIRONMENT (DTW_SYSTEM)    DTWSYS     ( OUT RETURN_CODE )
| ENVIRONMENT (HWS_LE)        DTWHWS     ( OUT RETURN_CODE )

```

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4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。71ページの『付録A. 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。

- 45ページの『メモリー管理用のユーティリティ関数』
- 46ページの『構成変数の管理用ユーティリティ関数』
- 46ページの『表操作のユーティリティ関数』
- 47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 5. 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
50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。
48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。
53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。

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 6. 構成ユーティリティー関数

ユーティリティー関数	説明
49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 7. 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
64ページの『dtw_table_New()』	テーブル・オブジェクトを作成します。
56ページの『dtw_table_Delete()』	テーブル・オブジェクトを削除します。
67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。
60ページの『dtw_table_GetV()』	表値を検索します。
69ページの『dtw_table_SetV()』	表値を設定します。
59ページの『dtw_table_GetN()』	表の列見出しを取り出します。
68ページの『dtw_table_SetN()』	表の列見出しを設定します。
66ページの『dtw_table_Rows()』	表内の現在の行数を検索します。
55ページの『dtw_table_Cols()』	表内の現在の列数を検索します。
63ページの『dtw_table_MaxRows()』	表内の最大許容行数を検索します。
65ページの『dtw_table_QueryColnoNj()』	列の列番号を取り出します。
54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。
62ページの『dtw_table_InsertRow()』	表に 1 つまたは複数の行を挿入します。
58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 7. 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
61ページの 『dtw_table_InsertCol()』	表に 1 つまたは複数の列を挿入します。
57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 8. 行ユーティリティ関数

ユーティリティ関数	説明
51ページの 『dtw_row_SetCols()』	行の幅を設定します。
52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. 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"
#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 (1/12)

```

/*-----*/
/*
/* 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 (2/12)

```

/*-----*/
/*
/* 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 (3/12)

```

/*-----*/
/* 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 (4/12)

```

/*-----*/
/*
/* 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 (5/12)

```

/*-----*/
/*
/* 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 (6/12)

```

/*-----*/
/*
/* 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 (7/12)

```

/*-----*/
/* 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 (8/12)


```

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

/*-----*/
/* 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 (9/12)

```

/*-----*/
/* 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 (10/12)

```

/*-----*/
/* 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 (11/12)

```

/*-----*/
/*
/* 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 (12/12)

付録B. 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	Lotus
DataJoiner	MVS
DB2	Net.Data
Domino	OS/2
IBM	OS/390
IMS	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. により例外的に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

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)。

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 などの Java 対応のブラウザを処理し、HTML ページのロードの際にロードされる。

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 (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。

BLOB. 2 進ラージ・オブジェクト (Binary large object)。

キャッシュ (cache). 最近アクセスされたデータが入るメモリーの型。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。

CLOB. 文字ラージ・オブジェクト (Character large object)。

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

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)。

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。

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. 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)。

ICS. インターネット接続サーバー (Internet Connection Server)。

ICSS. Internet Connection Secure Server。

インターネット (Internet). 国際パブリック TCP/IP コンピューター・ネットワーク。

インターネット接続サーバー (Internet Connection Server). IBM の無保護 Web サーバー。

Internet Connection Secure Server. IBM の保護付き Web サーバー。

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。言語環境によっては、REXX、Perl、および Oracle などの Net.Data と一緒に提供されるものもある。また、独自の言語環境を作成することもできる。

Live コネクション (Live Connection). 接続管理プログラムおよび Web サーバー API を処理する Net.Data 構成。Live コネクションを用いると、データベースの接続を再利用することができる。

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間にあるソフトウェア。異機種

混合のオペレーティング・システム全体で、異なるアプリケーション間の相互作用を管理する。 *Free Online Dictionary of Computing* より。

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。

Perl. 解釈済みプログラミング言語。

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。

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)。

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>。

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 37

dtw_lei フラグ 37, 42

一度に 1 行処理

dtw_getNextRow() 40, 41

dtw_lei フラグ 37

DTW_LE_CONTINUE 37

インターフェース関数

言語環境の説明 39

処理順序 39

dtw_cleanup() 42

dtw_execute() 40

dtw_getNextRow() 41

dtw_initialize() 40

受け渡し

パラメーター 37

変数 37

エラー条件 35

エラー条件メッセージ 37

[カ行]

記憶域

解放 37, 39, 48

割り当て 50, 51, 53, 67

dtw_lei フラグ 37

行操作のユーティリティー関数 47

行の最大数 63

クリーンアップ処理

異常条件のフラグ 42

異常条件用フラグ 37

処理後 40, 42

dtw_lei フラグ 37, 42

言語環境

インターフェース関数 39

インターフェース・テンプレート 71

概要 3

構成 42

構造体 36

作成 33

システム 26

初期化 39

処理後のクリーンアップ処理 40, 42

言語環境 (続き)

紹介 45

ステートメントの実行 40

フラット・ファイル・インターフェース 5

ユーティリティー関数 45

IMS Web 8

Java アプリケーション 15

Java アプレット 9

ODBC 19

Oracle 20

Perl 22

REXX 23

SQL 24

Sybase 24

Web 登録 27

構成、環境の 42

構成変数

管理用のユーティリティー関数 46

変数値の検索 49

構造体、言語環境

dtw_lei 36

dtw_parm_data 38

[サ行]

指示、記憶域の 50

システム

言語環境 26

変数の受け渡し 26

実行、言語環境ステートメントの 40

初期化、言語環境タスクの 39, 40

[タ行]

致命的エラー、dtw_lei フラグ 37, 42

テンプレート、言語環境 71

動的なメモリー割り当て 39

特記事項 83

[ハ行]

パラメーター

受け渡し 37, 38

指定 38

命名 39

parm_name 39

ヒープ、Net.Data 実行時 39

表

行の追加 54

削除 56

表 (続き)

新規の作成 64

ユーティリティ関数の操作 46

表値

検索 60

削除 56, 60, 69

割り当て 52, 69

表の作成 64

フラット・ファイル・インターフェース

機密保護に関する考慮事項 5

組み込み関数 6

言語環境 5

変数

受け渡し 37

解放 42

[マ行]

メモリー管理ユーティリティ関数 45

[ヤ行]

ユーティリティ関数

行操作 47

言語環境 45

構成変数管理 46

表操作 46

メモリー管理 45

dtw_free() 48

dtw_getvar() 49

dtw_malloc() 50

dtw_row_SetCols() 51

dtw_row_SetV() 52

dtw_strdup() 53

dtw_table_AppendRow() 54

dtw_table_Cols() 55

dtw_table_DeleteCol() 57

dtw_table_DeleteRow() 58

dtw_table_Delete() 56

dtw_table_GetN() 59

dtw_table_GetV() 60

dtw_table_InsertCol() 61

dtw_table_InsertRow() 62

dtw_table_MaxRows() 63

dtw_table_New() 64

dtw_table_QueryColnoNj() 65

dtw_table_Rows() 66

dtw_table_SetCols() 67

dtw_table_SetN() 68

dtw_table_SetV() 69

用語集 84

呼び出し、アプレットの 10

[ラ行]

列

削除 57

挿入 61

表内の数の指定 67

表内の合計の判別 55

列見出し

記憶域の割り当て 51, 67

検索 59

削除 56, 59, 68

名前の割り当て 68

列番号の戻し 65

D

DB2、SQL 言語環境 24

dtw_ インターフェース関数 39

dtw_ 構造体 36

dtw_ ユーティリティ 45

DTW_APPLET 9

DTW_FFI 5

DTW_IMS 8

DTW_JAVAPPS 15

dtw_lei

構造体 36

フィールド

関数名 36

default_error_messages 37

exec_statement 37

flags 36

le_opaque_data 37

parm_data_array 37

row 38

DTW_LE_CONTINUE 41

DTW_ODBC 19

DTW_ORA 20

dtw_parm_data

構造体 38

フィールド

parm_descriptor 38

parm_name 39

parm_value 39

DTW_PERL 22

DTW_REXX 23

DTW_SQL 24

DTW_SYB 24

DTW_SYSTEM 26

DTW_WEBREG 27

E

ENVIRONMENT ステートメント

新しい言語環境用 42

構文 42

例 44

EXEC ステートメント、dtw_lei フラグ 37

F

FUNCTION ブロック

ステートメントの実行 40

name 36

I

IMS Web

言語環境 8

スタジオ・ツール 9

J

Java アプリケーション

関数の作成 17

言語環境 15

特殊構成 17

呼び出し 18

CLIETTE の作成 17

Java アプレット

クラス 14

言語環境 9

作成 10

タグの生成 10

呼び出し 10

例 13

O

ODBC

言語環境 19

マクロ・ファイルの SQL ステートメント 19

Oracle

アクセス 20

言語環境 20

ストアド・プロシージャ 20

特殊構成 20

P

parm_data_array 構造体名の割り当て 37

Perl

言語環境 22

スクリプト内の Net.Data 変数 22

R

REXX

言語環境 23

変数置換 23

rows

許容最大数の戻り 63

現在数の検索 66

削除 57, 58

挿入 62

追加 54

幅の割り当て 51

戻り 38, 40, 41

dtw_getNextRow() インターフェース関数 38

S

SQL

言語環境 24

サポートされるステートメント 24

DataJoiner の使用 24

Sybase

アクセス 24

言語環境 24

特殊構成 24

ラージ・オブジェクト 24

W

Web 登録

言語環境 27

説明 28



Printed in Japan

SB88-7368-00

